Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de múltiples interfaces para el control remoto de un vehículo y del sistema de visión embarcado

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Viacheslav Brydko

Tutor: Juan Vicente Capella Hernández

2014-2015

Resumen

En el trabajo adjunto se presenta un proyecto de diseño e implementación de un sistema automatizado de reconocimiento de matrículas de vehículos controlado remotamente. Dicho sistema tiene como objetivos principales la localización y el reconocimiento de un vehículo conforme a su placa de matrícula en un espacio amplio, además de disponer de otras funcionalidades adicionales, como la vigilancia, por ejemplo. El proceso de reconocimiento se realiza mediante una plataforma móvil, controlada remotamente, con capacidad de visión por ordenador y transmisión de video en tiempo real al puesto de control.

Para la comodidad del usuario, se ofrecen distintas formas de controlar la conducción de la plataforma, que pueden ser usadas en distintas condiciones del terreno o según las preferencias personales del operador. Dichas opciones son compuestas por los siguientes periféricos de entrada: control por mando, basado en movimientos y basado en control táctil. En el primer caso se trata de un controlador de videojuegos y los dos siguientes son provistos por un teléfono inteligente.

Las señales de control se transmiten al sistema instalado a bordo del vehículo, que realiza tres funciones: el envío del video capturado por la cámara montada en el vehículo al operador en tiempo real, reconocimiento de matrículas y reenvío de señales de control recibidas al controlador empotrado encargado de mover el vehículo.

En el apartado *hardware,* el vehículo es formado por un chasis de un coche de radiocontrol con una placa computadora (Raspberry Pi 2) con los siguientes periféricos conectados: módulo WiFi, que proporciona conectividad, una cámara conectada al puerto CSI que se encarga de transmitir el video capturado, otra cámara conectada al puerto USB que se encarga del reconocimiento de matrículas y, finalmente, un microcontrolador (Arduino), que se encarga de controlar el motor eléctrico propulsor y el servomecanismo que sirve para contolar la dirección. Todo el sistema es alimentado por dos baterías de ion-litio que constituyen tres fuentes de energía independientes que alimentan por separado a los componentes que más energía requieren: Raspberry Pi y sus periféricos, el motor principal y el servomecanismo.

Este hardware es gobernado por una serie de programas que se ejecutan en la propia placa computadora, el microcontrolador conectado a esta y un ordenador conectado a la misma red. Estos programas intercambian mensajes para transmitir información y notificar distintos cambios de estado en el que se encuentra el sistema.

Palabras clave: Raspberry Pi, Arduino, móvil, tiempo real, visión por ordenador.

Abstract

The following work presents a design and implementation of an automated system with optical vehicle license plate recognition and remote operation capabilities. The objectives of this system are location and recognition of vehicles in wide spaces, as well as offering other features, such as surveillance, for example. The optical recognition is performed by a remotely controlled mobile platform with computer vision capabilities and live video streaming to the operator screen.

Several control options are offered for user convenience, which may be useful in different environments or could being chosed according to user´s personal preferences. These options are composed by the following input peripherals: controller-based, motion-based and touch-based controls. In first case the peripheral is a high performance gamepad and latter options are provided through software executed on a smartphone.

The control signals are transmitted to the system installed on board of a vehicle, which performs three functions: real-time live streaming of the video captured by the camera mounted on top of the vehicle to the user, optical license plate recognition and forwarding of control signals to the embedded microcontroller, which is responsible of the movement of the vehicle.

The hardware part of the project is composed by the radio-controlled car chassis with a single-board computer (Raspberry Pi 2) with the following peripherals: WiFi module, for communications, main camera connected to the CSI port for video streaming, secondary camera connected to the USB port for the license plate recognition and a microcontroller (Arduino) which is used for control the electric motor, which propels the vehicle and the servomotor, which provides the vehicle with steering. The system is powered by two ion-lithium batteries, which form three independent energy sources used by the most energetically demanding components: Raspberry Pi and its peripherals, the main electric motor and the servomotor.

The hardware is controlled by a set of programs which run on the main computing board, connected microcontroller and a computer connected to the same network as the Raspberry Pi. This programs exchange messages in order to transmit information and notify each other about the current system state.

Keywords: Real-time, computer vision, Raspberry Pi, optical recognition, Arduino.

Tabla de contenidos

1. Introducción 7
Motivación 7
Objetivos 8
Objetivos Personales 8

2. Tecnologías relacionadas 9
Raspberry Pi 2 9
Arduino 12
Mando PC / Xbox 360 14
Qt 14
OpenCV 15
Python 16
Android 16

3. Sistema propuesto 17
Raspberry Pi 17
Periféricos de Raspberry Pi 21
Arduino 24
Periféricos Arduino 24
Alimentación 26
Chasis 27
Software 28
Software de escritorio 29
Software Android 33
Software embebido 34
Software Raspberry Pi 34

Software Arduino....36

Reconocimiento de matrículas (ANPR) 37

Obtención de la foto del vehículo....37

Primera solución....38

Segmentación de la imagen para la obtención de la matricula....38

Reconocimiento de la matrícula....41

Reconocimiento óptico de caracteres....42

Conclusiones y análisis de esta posible solución....42

Segunda solución....42

Partes....43

Inicialización....44

Obtención de la imagen....45

Segmentación....45

OCR....46

4. Análisis y conclusiones 49

5. Bibliografía 50

Anexo I – JNI 51

1. Introducción

A continuación se expone el proceso y el resultado del diseño e implementación de un sistema combinado, capaz de realizar operaciones a control remoto con el especial énfasis en la usabilidad y la eficacia. El resultado es la combinación de distintas tecnologías capaces de funcionar al unísono, intercambiando datos, cumpliendo con los objetivos y reduciendo las limitaciones que puedan asociarse a su uso. Al mismo tiempo, se ha hecho un especial hincapié en la experiencia del usuario, ofreciéndole la posibilidad de elegir entre múltiples opciones disponibles y permitiéndole un control total sobre el sistema en tiempo real.

Motivación

Este proyecto combina dos funcionalidades básicas: el reconocimiento óptico de matrículas y la movilidad del sistema, siendo controlado por distintos periféricos de entrada. Las áreas donde se pueden aplicar esta combinación de características son variadas y con distintos objetivos. Se pueden distinguir al menos tres aplicaciones donde una solución como la propuesta puede encontrar su uso.

1. En primer lugar se encuentran los controles de aparcamiento. En una zona azul un controlador de la ORA puede recurrir a un sistema móvil de reconocimiento de matrículas controlado remotamente para comprobar la validez de los tickets de cada vehículo en vez de comprobarlos a mano presencialmente.

2. En segundo lugar, este sistema puede proporcionar a los vigilantes de los parkings privados servicios de vigilancia y control de accesos, gracias a su alta movilidad y la capacidad de transmisión de video en tiempo real.

3. Por último, los cuerpos de seguridad pueden hacer uso de un sistema similar para encontrar vehículos buscados, los que incumplan la normativa vigente o a titulares de los mismos. Esto puede ser posible gracias a la integración con los servicios telemáticos ofrecidos por la DGT que permiten a los agentes acceder al registro de vehículos nacional para comprobar datos como, por ejemplo, la caducidad de la ITV, el seguro obligatorio, el estado del vehículo (secuestrado o dado de baja) así como de los datos del titular (nombre, domicilio). Dichos servicios se llaman ATEX y son usados en exclusiva por la Policía, Guardia Civil y los ayuntamientos, entre otros organismos.

En caso de dotar a la plataforma de una capacidad de hacer uso de servicios descritos arriba su eficacia crece sustancialmente, ya sea usada como estación móvil o fija. Cualquiera de las aplicaciones descritas tiene su ámbito en un contexto, pero además,

estos pueden combinarse entre sí, e incluso añadirse capacidades nuevas conforme los requerimientos existentes.

Objetivos

Este proyecto tiene como finalidad ofrecer una solución integrada, capaz de llevar a cabo múltiples operaciones simultáneamente de una forma trasparente al usuario eficazmente. Una forma de cumplir con esta máxima, es ofrecer alta movilidad, gran autonomía, elevada velocidad de tratamiento de imágenes y reducidas latencia de transmisión de video y coste total de la plataforma. En caso de fallar en cualquiera de los objetivos anteriores todo el sistema deja de ser usable y en vez de ser una solución viable, se convierte en un producto poco útil.

El objetivo principal del sistema es el reconocimiento óptico de matrículas, al tratarse de un problema complejo desde un punto de vista computacional, pero que al mismo tiempo ofrece a los usuarios disfrutar de unas capacidades de tratamiento de imágenes aplicado a su entorno laboral, que les pueda permitir realizar sus tareas más rápida y cómodamente. Como apoyo a la plataforma y con tal de facilitar a los usuarios su trabajo en caso de que la plataforma no sea usable, también se considera la opción de instalar el software responsable del reconocimiento de matrículas en un teléfono inteligente con sistema operativo Android. De esta forma el factor limitante asociado al uso del sistema se reduce, al operar con una solución autónoma de reconocimiento óptico de matrículas.

El objetivo secundario es la posibilidad de operar la plataforma a control remoto, pues uniendo esta capacidad a la anterior, el producto resultante puede realizar muchas más operaciones en más contextos operativos, además de enfocarse a distintos grupos de usuarios.

Objetivos personales

A nivel personal, los objetivos perseguidos son, sobretodo, aplicar los conocimientos adquiridos a los largo de los años de formación en esta escuela, y compensando con investigación, uso de documentación y trabajo la falta de conocimientos específicos sobre distintas tecnologías, tales como OpenCV o mecanismos usados para la comunicación entre JVM y librerías dinámicas nativas escritas en C++. Adicionalmente, la implementación del sistema propuesto se lleva a cabo en distintas plataformas: móviles (Android), embebidas (Arduino), computacionalmente reducidas (Raspberry Pi) y de escritorio. Además se hace uso de distintos lenguajes de programación en la *suite* de *software* que maneja el hardware anterior, tales como C++, Java, Python, Processing y Bash.

El desarrollo de un proyecto tan completo como este, que aúna distintas tecnologías, tanto a nivel hardware como software, representando la necesidad de superar limitaciones y problemas que se encuentran en el camino y el hecho de superarlos satisfactoriamente no hace más que subrayar la solidez de la formación recibida.

2. Tecnologías relacionadas

A continuación se nombran las tecnologías más relevantes usadas en el proyecto. Estas se componen de componentes *hardware* y *software*. En esta sección se discuten los motivos de su elección y las características destacables de los mismos.

Raspberry Pi 2

Como parte lógica principal, encargada de procesar los datos, hacer uso de las comunicaciones y controlar indirectamente los periféricos de bajo nivel (motor y servomecanismo) se utiliza una placa computadora (o SBC) Raspberry Pi 2.

Figura 1: Raspberry Pi 2 Modelo B+

Esta elección está motivada por el hecho de que esta unidad cumple con las siguientes características: bajo coste, cantidad de software disponible, alto volumen de documentación disponible, bajo consumo energético, tamaño y peso reducidos, desarrollo constante y muchos aportes por parte una comunidad de desarrolladores

muy activa. Estas premisas son válidas tanto para la primera generación de estas placas como para la siguiente, la actual.

En la siguiente tabla se puede apreciar las diferencias entre las CPUs de la primera y segunda generación de las Raspberry Pi:

	ARM Cortex-A7	**ARM1176JZF-S**
Raspberry Pi SOC	Broadcom BCM2836	Broadcom BCM2835
Arquitectura	ARMv7-A	ARMv6KZ
Número de núcleos	4 núcleos	1 núcleo
Juego de instrucciones	ARMv7	ARMv6
Instrucciones de 16 bits	Thumb-2	Thumb
Multiprocesamiento simétrico	2-4 núcleos	NO
Bús del sistema	128 bits	32 bits
Cache *Snoop*	L1 y L2	NO
Multihilo	NO	NO
Pipeline	8 niveles	8 niveles
Tipo de *pipeline*	2 instrucciones (parcial)	1 instrucción
Ejecución	en orden	en orden
Unidad de gestión de memoria	40 bits	32 bits
Unidad de coma flotante	VFPv4	VFP11
Unidad de coma flotante segm.	SI	SI
NEON SIMD	SI (64 bits)	NO
Cache L1 comandos	32 KB	16 KB
Cache L1 datos	32 KB	16 KB
Cache L2	512 KB, compartida	NO
Proceso tecnológico	40 nm	90 nm
DMIPS / MHz	1.95 DMIPS / MHz	1.2 DMIPS / MHz
Consumo	0.1 mW / MHz	0.2 mW / MHz
Año aparición	2011	2003

Tabla 1: Comparativa entre SOCs de Raspberry Pi de primera y segunda generación

Para la implementación de este diseño se utiliza una unidad de segunda generación debido a que la naturaleza de este proyecto exige la disponibilidad de una serie de características que el modelo anterior no posee. Principalmente, estas exigencias vienen dadas por el hecho de poseer una elevada potencia de cómputo, concurrencia y capacidad vectorial. Viendo estos requisitos en detalle se puede observar que:

- En primer lugar, la necesidad de contar con una CPU potente viene dada por el elevado número de operaciones exigentes que se realizan: procesamiento de imágenes, transmisión de video, comunicaciones y necesidad de disponer de

una pequeña reserva de potencia en caso de añadir nuevas funcionalidades en un futuro, todo esto sin contar el *overhead* existente del sistema operativo subyacente.

- En segundo lugar, estas operaciones se realizan en paralelo, por lo que contar con una capacidad concurrente es imprescindible. Para lograrlo es necesario disponer de un sistema multinúcleo, capaz de ejecutar procesos paralelamente de forma simultánea.

- Por último el hecho de disponer de capacidades de computación vectorial acelera las operaciones de procesamiento de imágenes, aumentando notablemente el rendimiento y haciendo que la experiencia de usuario sea óptima.

La principal ventaja del segundo modelo frente al primero es el uso de núcleos ARM Cortex-A7 con micro arquitectura ARMv7-A frente a antiguos núcleos ARM11 con micro arquitectura homónima. La diferencia entre ambos procesadores radica en el conjunto de instrucciones usado y características que ofrece cada uno de ellos. En primer lugar el conjunto de instrucciones usado por la Raspberry Pi de segunda generación es ARMv7 contra ARMv6 usado en modelos de primera generación. En la actualidad ARMv7 es el conjunto de instrucciones para procesadores ARM más popular del mundo. Esto se debe a la introducción de tres características fundamentales respecto a ARMv6, manteniendo el mismo modelo de programación.

Las tres características fundamentales introducidas en ARMv7 son: ThumbEE, NEON y VFPv4. El siguiente gráfico ilustra la evolución de las arquitecturas de ARM:

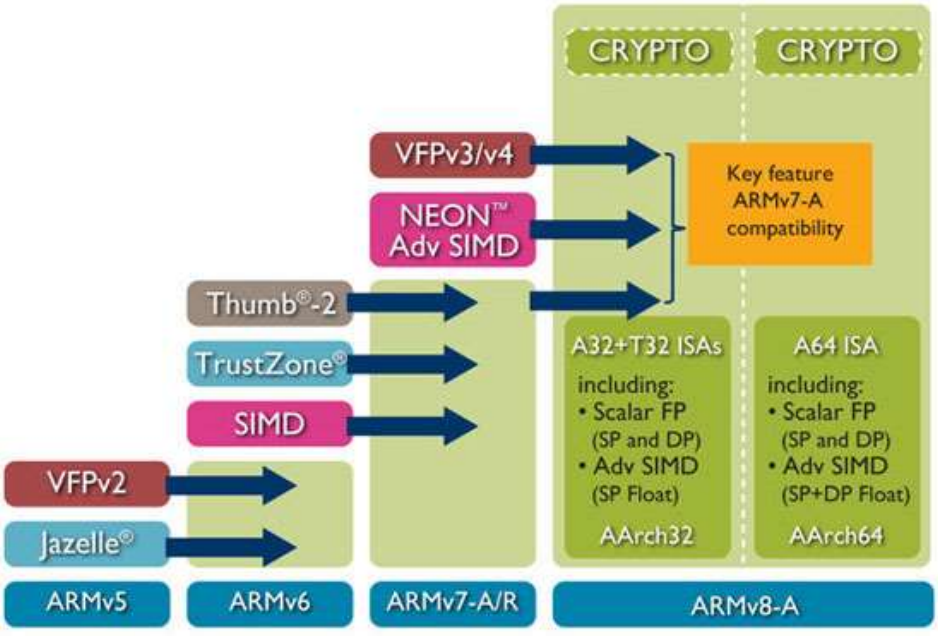

Figura 2: Evolución de las arquitecturas ARM

- ThumbEE es un modo especial en el que puede ponerse la CPU al ejecutar ciertas instrucciones. Por ejemplo, cuando se ejecutan operaciones ARM, la CPU se encentra en el estado ARM. Cuando el código a ejecutar es generado dinámicamente, como puede ser el caso de lenguajes con capacidades JIT o AOT, tales como Java o C#, entre otros, la CPU puede seleccionar este modo y ejecutar las instrucciones *bytecode* de estos lenguajes directamente por el hardware de la CPU con la consiguiente mejora de rendimiento y reducción de consumo.

- La tecnología NEON es la implementación de la tecnología SIMD en los procesadores ARM, disponible como una ampliación de la arquitectura existente. En ciertos procesadores su inclusión es opcional, pero, actualmente la inmensa mayoría de CPUs ARM disponen de ella. Esta tecnología permite al procesador aplicar una misma operación sobre un conjunto de datos. Se compone del conjunto de instrucciones propio, hardware dedicado (*pipeline* propio) y un conjunto de registros exclusivo. En este proyecto se hace uso de esta tecnología para acelerar notablemente el procesamiento de imágenes en la fase de detección óptica de matrículas. Convenientemente, todo software ha de ser compilado con el *flag* del compilador correspondiente, que indica el uso de esta característica.

- VFP (o *Vector Floating Point*) es un coprocesador que extiende la arquitectura ARM con una unidad de computación de bajo coste de números de coma flotante de simple y doble precisión. El hecho de disponer de esta característica también beneficia el procesamiento de imágenes, al reducir el tiempo invertido en cálculos computacionalmente costosos.

Arduino

Debido a ciertas limitaciones de la Raspberry Pi, esta no puede usarse para controlar todos los periféricos del sistema directamente, más concretamente el motor y el servomecanismo. Esto se debe, fundamentalmente, a dos limitaciones, tales como el hecho de que Raspberry Pi únicamente trabaja con TTL 3V3 y que dispone de un solo canal PWM.

Esto significa que no se puede conectar periféricos que utilicen TTL 5 V porque la placa no los tolera sin un convertidor de nivel, capaz de convertir los niveles lógicos TTL 3V3 a 5 V y por otro lado, solo se puede controlar o el servomotor o el motor principal usando PWM. El uso de un microcontrolador evitaría estos inconvenientes, facilitando el desarrollo, reduciendo la complejidad además de hacer el sistema más escalable en términos de componentes que puedan ser conectados en un futuro.

La elección del microcontrolador cayó sobre una placa Arduino Nano al disponer de un puerto USB integrado, ser una solución de bajo coste, consumo reducido y un factor de forma minimalista.

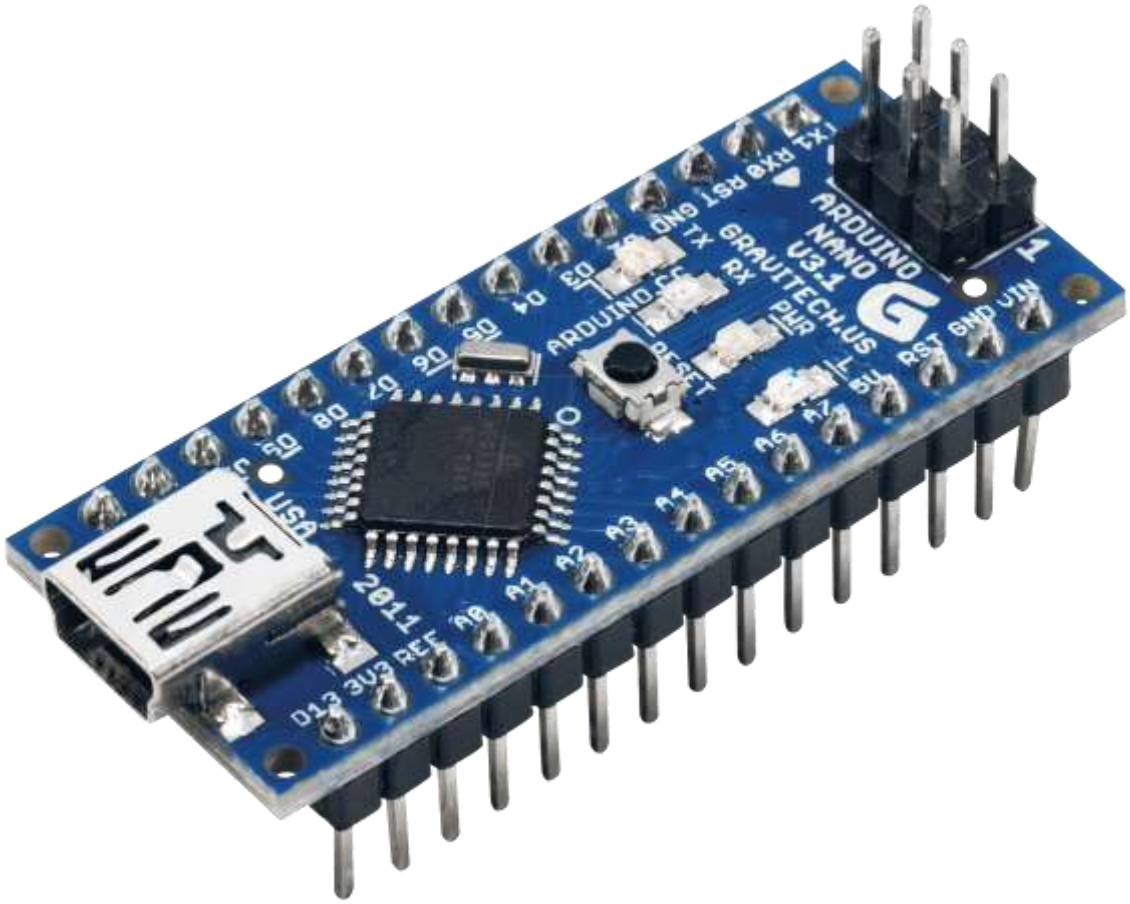

Figura 3: Arduino Nano 3.0 (ATmega328)

Para reducir las limitaciones de la Raspberry Pi comentados anteriormente, se ha decidido conectar un microcontrolador Arduino a la Raspberry Pi y usar la conexión serie vía USB, existente entre las dos placas para el intercambio de mensajes. De esta forma cuando se desea realizar una acción sobre algún componente, se le manda un mensaje a Arduino para que lo haga. De forma equivalente, Arduino puede notificar a la Raspberry Pi sobre el estado de distintos sensores o lecturas de los mismos, en caso de ser necesario, vía mensaje, a través de la conexión serie.

Esta solución permite conectar muchos componentes distintos y controlarlos sin preocuparse por posibles daños que se le puedan causar a la Raspberry Pi o estar limitado por el número de canales PWM. Los daños a las placas pueden ocasionarse por superar la corriente máxima aplicada a un pin dado o al conjunto de los mismos. En caso de la Raspberry Pi, el máximo es de 16mA por pin y un máximo de 51 mA para todos en conjunto. Arduino, al tratarse de un microcontrolador es mucho más tolerante y soporta corrientes de salida de 40 mA por pin y un total de 200 mA para el conjunto

de pines. Además, Arduino admite voltajes de entrada de entre 5 V y 20 V, siendo una elección más flexible en muchos diseños de sistemas electrónicos, mientras que Raspberry Pi exige una alimentación mucho más estricta, 5 V siempre y >1 A en la mayoría de los casos.

Por último, Arduino dispone de una serie de pines analógicos, capaces de operar con sensores del mismo tipo. Esto es posible gracias a que este incorpora un conversor analógico-digital de 10 bits de precisión. En caso de la Raspberry Pi, las operaciones con componentes analógicos no son posibles sin un hardware especializado.

Viendo las limitaciones de la Raspberry Pi en términos de manejo de distintos componentes, la inclusión de un microcontrolador especializado en operaciones de bajo nivel elimina dichas limitaciones, permitiendo operar sobre múltiples componentes y añadir más piezas con distintas capacidades en futuras evoluciones del sistema. La combinación existente entre las dos placas crea una separación entre el hardware y la lógica, tanto a nivel físico, como a nivel lógico. La placa principal, Raspberry Pi es la responsable de la ejecución de software de reconocimiento óptico de matrículas, gestionar las comunicaciones y coordinar el trabajo de componentes subyacentes controlados por Arduino. Finalmente, Arduino se encarga de ejecutar el programa que controla directamente los componentes conectados a él, conforme los comandos recibidos de la Raspberry Pi.

Mando PC / XBOX 360

Para el periférico de control, se eligió el mando inalámbrico de Microsoft usado en Xbox 360 / PC. Este periférico proporciona una ergonomía, precisión, autonomía y un elevado número de controles digitales y analógicos, que lo hacen idóneo como interfaz de hardware para la interacción persona-computador en el contexto propuesto. Quizás la característica más destacable de este dispositivo es la gran precisión de los controles analógicos, que son los más usados. Distinguiendo entre dos tipos de los mismos, se pueden encontrar el joystick y los gatillos. La elevada precisión viene dada por la resolución de los controles analógicos comentados anteriormente, siendo de 16 bits para los joysticks y de 8 bits para los gatillos.

Qt

Para el desarrollo de software que ejecuta el operario para enviar los datos de control al vehículo y recibir las matriculas reconocidas, se ha hecho uso de un *framework* multiplataforma de desarrollo de aplicaciones en C++, aunque distintos interfaces (o *bindings*) para otros lenguajes de programación también están disponibles.

La elección de un entorno de desarrollo nativo (o compilado como es el caso de C++) viene dada, principalmente, por la necesidad de acceder a la API del mando de la Xbox

360, llamada XInput, que forma parte del SDK de DirectX y que son ofrecidos nativamente por sistemas operativos Windows.

Además, Qt implementa el patrón de diseño "Observador" (llamado *signal-slot* en esta implementación), que permite notificar cambios de estado de un objeto a muchos otros, lo que hace el desarrollo de sistemas orientados a eventos muy cómodo. Dado que el programa en cuestión, debido a su naturaleza, es el ejemplo perfecto de un desarrollo orientado a eventos, esta herramienta es idónea para su implementación.

Adicionalmente, Qt proporciona una capa de abstracción de red muy potente y versátil, que permite crear aplicaciones que usan comunicaciones de alto y bajo nivel, tales como HTTP, FTP o sockets, entre otros ejemplos.

Debido a la naturaleza concurrente de esta aplicación en concreto, se ha hecho uso de la capacidad de Qt de crear y gestionar hilos independientemente del sistema usado.

Finalmente, se ha hecho uso de otra característica destacable de Qt, que es la creación de interfaces de usuario. Actualmente Qt dispone de dos mecanismos de creación de interfaces: basado en *widgets* y declarativo. En primer caso se usan los componentes propios de cada sistema operativo, pero de forma abstracta, hecho necesario para la portabilidad de aplicaciones entre distintos sistemas operativos. La ventaja de este método radica en el *look & feel* nativo del sistema operativo en el que se ejecuta el programa. Como desventaja se puede poner la posibilidad de corrupción del *layout* en distintos SO debido a tamaños y formas variables de los componentes. Por otro lado la nueva forma propuesta por Qt de diseñar interfaces de usuario es declarativa, donde toda la interfaz se diseña usando técnicas de diseño web en vez de hacer uso de componentes ofrecidos por el SO. El resultado final es una interfaz que siempre es la misma sin importar la plataforma software sobre la cual se ejecuta. Este planteamiento permite diseñar interfaces reutilizables para distintos tamaños de pantalla, resoluciones y densidades de pixeles.

OpenCV

Para la parte del reconocimiento de matrículas, que implica procesamiento de imágenes, se ha hecho uso de la popular librería de visión por ordenador y tratamiento de imágenes OpenCV, dado que esta proporciona un amplio conjunto de herramientas y algoritmos, tanto exclusivos al procesamiento de imágenes, como de propósito general. Además, es compatible con muchos sistemas operativos existentes y es libre de ser usada en proyectos comerciales y *open-source* al estar sujeta a una permisiva licencia BSD. La estructura de OpenCV divide los componentes que la forman en módulos, tales como *core*, *imgproc*, *video* y *contrib*, entre otros. Cada uno de estos

módulos contiene estructuras de datos, funciones y algoritmos relativos a un estrecho margen de especialización indicado por el nombre del propio modulo.

Python

Al tratarse de una distribución Linux, completamente compatible con todos los repositorios Debian *armhf*, El *fork* del mismo para Raspberry Pi, Raspbian incluye la plataforma Java en su versión JVM 1.8 de Oracle. Esto permite ejecutar todo tipo de software que pueda ser compilado para la JVM (Java, Jython, Groovy, Scala u otros). Durante el proceso de diseño se consideró usar Java como software encargado de recibir datos de control y establecer la conexión serie con Arduino y, finalmente enviar al usuario las matriculas reconocidas. Tras un breve análisis de este tipo de solución, el uso de Java se deshecho en favor de Python.

El motivo principal es la complejidad a la hora de establecer la conexión serie. La implementación del *runtime* de Java, JVM, no tiene un mecanismo para tal fin, por lo que se usa una librería externa, llamada *TxRx*. El hecho de usar una librería externa no representa un problema en sí, pero en este caso la limitación viene dada por la implementación de comunicación serie a nivel del sistema operativo subyacente, por lo que la librería *TxRx* ofrece una solución hibrida: una librería nativa para el SO y un JAR, para exponer las nuevas funcionalidades a la plataforma Java. En caso de Python, la comunicación serie no tiene mayor complejidad que declarar el puerto serie provisto por el sistema operativo, abrirlo y realizar operaciones de lectura / escritura sobre el mismo.

Otra de las ventajas que ofrece el uso de Python es la facilidad de trabajo con comunicaciones, pudiéndose establecer operaciones que se realizan a través de la red en pocas líneas de código.

Finalmente, las transformaciones de datos en Python también son muy sencillas de realizar. Operaciones como *parsear* JSON no requieren hacer uso de librerías externas, como si es caso de Java y son muy comprensibles.

Android

Para la implementación de las interfaces de control vía software en un dispositivo móvil se decidió por usar una unidad con el sistema operativo Android. La elección de esta plataforma viene motivada por su enorme popularidad en el mercado y debido a ello a la cantidad de información y documentación disponible.

Además, el lenguaje de programación principal del SDK de Android es Java, por lo que la librería estándar hereda toda la potencia de esta plataforma, extendiéndose con estructuras de datos y construcciones propias de desarrollos para dispositivos móviles implementados por Google. El hecho de usar Java viene motivado por la necesidad de poder disponer de las aplicaciones producidas para versiones de Android ejecutados en

distintos tipos de *hardware*, tales como procesadores ARM, Intel o MIPS. En las versiones anteriores de Android, estas aplicaciones se compilaban a *bytecode,* que posteriormente se ejecutaba en una máquina virtual especial, adaptada a un entorno de ejecución de capacidades computacionales y energéticas limitadas llamado Dalvik. En las últimas versiones de este sistema operativo, las aplicaciones se compilan AOT durante la instalación, prescindiendo de esta forma de la capa de software intermedia, como lo fue Dalvik.

Finalmente, usar Java como lenguaje de desarrollo permite aprovechar la enorme cantidad de código y librerías de terceros existentes en Internet. El uso de estos recursos permite reducir el tiempo de desarrollo y aumentar la calidad del código. El ejemplo más popular lo constituye el uso de librerías Apache Commons de Apache Software Foundation para Java.

3. Sistema propuesto

El sistema propuesto está compuesto por dos partes diferenciadas: hardware y software. A continuación se describe la parte hardware y finalmente, las funcionalidades de la parte software se asocian con las capacidades de la parte hardware.

Raspberry Pi

La preparación de la placa lógica principal en el apartado de hardware no conlleva más operaciones que conectar sus periféricos a sus puertos correspondientes y conectarla a su propia fuente de alimentación. En el apartado de software, sin embargo se requiere realizar una serie de operaciones enfocadas a mejorar el rendimiento, estabilidad e incluso, realizar configuraciones que puedan hacer posible el despliegue del software de reconocimiento óptico de matrículas.

1. El primer paso es pasar de la versión actual estable de Debian a una versión más nueva, aunque inestable y no disponible públicamente. El motivo principal de este cambio es la necesidad de poder disponer de la suite de compiladores GCC más moderna (4.9 contra 4.6 de la versión anterior). Esto es necesario para poder compilar OpenCV con el módulo de texto opcional, así como el propio programa, que hace uso de estas dependencias. El *toolchain* del compilador GCC 4.6 de Raspbian basado en Debian 7 'Wheezy' no fue capaz de compilar el código fuente de OpenCV, por lo que se tuvo que cambiar a una versión más reciente de Raspbian 8 'Jessie' que incluye la versión 4.9 con soporte del estándar C++11 capaz de compilar el paquete de software dado. Adicionalmente, todos los paquetes software instalados con la instalación de Raspbian 'Jessie'

son *armhf*, es decir diseñados para ser usados con procesadores ARMv7, capaces de sacar partido a las características hardware de este tipo de procesadores descritos anteriormente (particularmente VFP). La versión anterior de Raspbian da soporte a las dos generaciones de la Raspberry Pi y, por tanto, es compatible con ambas sacrificando el rendimiento o uso de características exclusivas de ARMv7 en favor de la compatibilidad.

2. Dado que Raspberry Pi es un ordenador de pocos recursos y la parte de reconocimiento óptico de matrículas es un problema computacionalmente exigente, aumentando la velocidad de procesamiento, (también llamado *overclock*) se puede aumentar la velocidad del trabajo del software proporcionalmente. Para poder realizar el *overclock* en una placa Raspberry Pi se requiere editar el archivo `/boot/config.txt` introduciendo valores como estos:
    - arm_freq=1000
    - over_voltage=0
    - core_freq=500
    - sdram_freq=483

estos establecen la frecuencia del procesador en 1000 MHz, frecuencia de la cache L2 en 500 MHz, frecuencia de la memoria RAM en 483 MHz y no sube el voltaje al que trabaja el procesador. Unos valores de *overclock* más agresivos requieren modificar el voltaje del procesador, lo que implica un mayor consumo y calentamiento, pero en el diseño propuesto este paso no es necesario. Los valores mencionados contrastan con los valores por defecto de la Raspberry Pi de 900, 250 y 450 MHz, respectivamente, proporcionando un incremento de velocidad de cómputo de entre 20% y 40%, dependiendo de las instrucciones ejecutadas. Esta subida de rendimiento se debe a que el mayor cambio es que se produce es en la velocidad de la cache L2.

3. El siguiente paso en la preparación de la Raspberry Pi es la compilación de OpenCV desde el código fuente del repositorio oficial. Aunque Debian posee paquetes estables disponibles para ser instalados, estos no pueden ser usados al no disponer de módulos necesarios. Tal y como se explicó en el apartado de Tecnologías relacionados, la estructura de OpenCV está formada por módulos, tales como *core*, *highgui*, *features2d* y muchos otros. En caso de instalar la versión estable, se obtendrá acceso a los módulos estables. El software de reconocimiento de matrículas usado en este proyecto hace uso de un módulo experimental, llamado *text* que no forma parte de la distribución de OpenCV, sino que se ofrece en un repositorio aparte llamado *opencv_contrib* como desarrollos aparte compatibles y usables, pero en estado de pruebas y validación para poder formar parte de OpenCV en un futuro. Por esta razón, si se desea disponer de las funcionalidades ofrecidas por el modulo texto se requiere compilar el código fuente de todos los módulos a usar (incluyendo otros módulos, que son sus dependencias).

   El proceso de compilación es sencillo si se hace uso de la herramienta de edición de archivos de script CMAKE, usados por OpenCV. Para poder compilar basta con seguir los siguientes pasos:

- descargar el código fuente de OpenCV en una carpeta,
- descargar el código fuente de los módulos extra en otra carpeta,
- ejecutar *cmake-gui* (herramienta de edición de scripts CMAKE)
- indicarle al programa la ruta del código fuente de OpenCV
- indicar la ruta de la carpeta donde se va a compilar
- configurar el script indicando las opciones deseadas
- generar los scripts en la carpeta de destino
- ir a la carpeta de destino y ejecutar `make -j4 && make install`

Durante el proceso de configuración es necesario marcar o desmarcar ciertas opciones importantes, tales como la localización de la carpeta con los módulos opcionales, soporte de librerías de terceros, tales como CUDA u OpenCL. Una configuración incorrecta puede resultar en un fallo al compilar o en caso de completarse, puede no tener la estabilidad deseada u otros problemas. El proceso de configuración es ilustrado en la siguiente imagen:

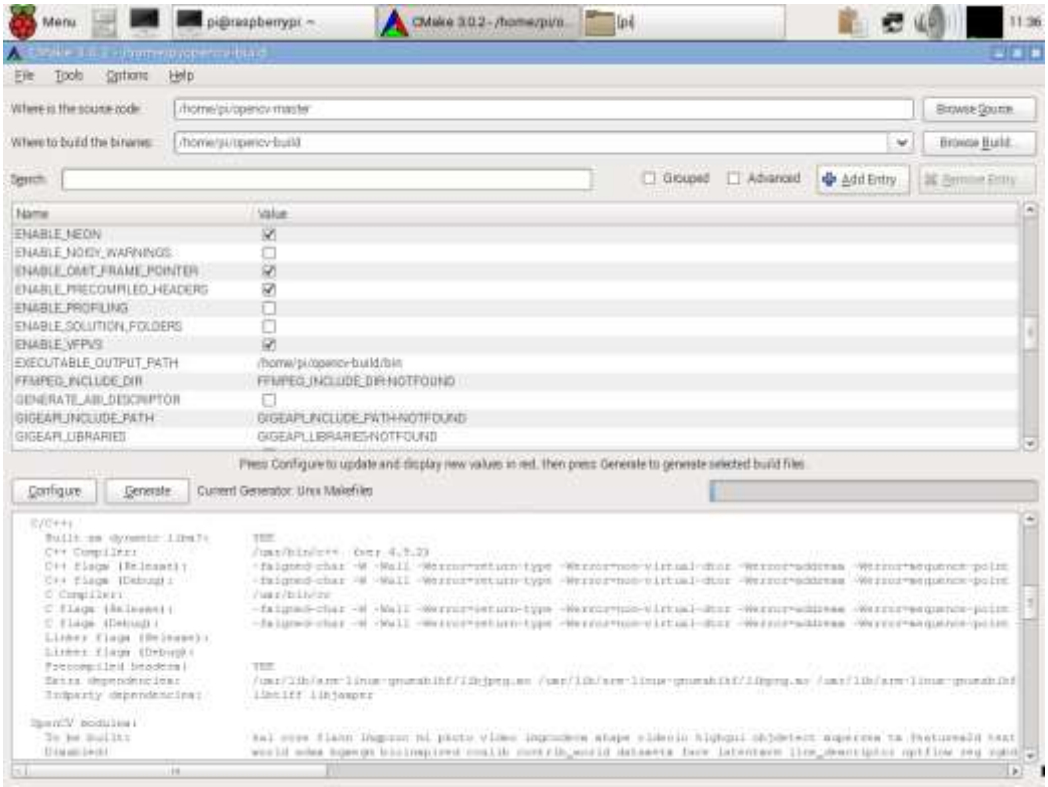

Figura 4: Generación de makefile usando cmake-gui

Una de las opciones más importantes a marcar es el uso de capacidades NEON y VFPv3 ofrecidos por la CPU Cortex-A7 con micro arquitectura ARMv7 de la Raspberry Pi 2. Sus ventajas están explicadas en el apartado anterior. Al marcarla, el *build* resultante de OpenCV podrá sacar todo el partido al ofrecido el hardware.

4. El último paso en el proceso de configuración es instalar Code::Blocks, un IDE *open source* enfocado a desarrollos de software en C++. Este está disponible en los repositorios de Debian 'Jessie'. Una vez instalado, se procede a la creación y configuración del proyecto para la habilitar el uso de las librerías OpenCV recién compilado.

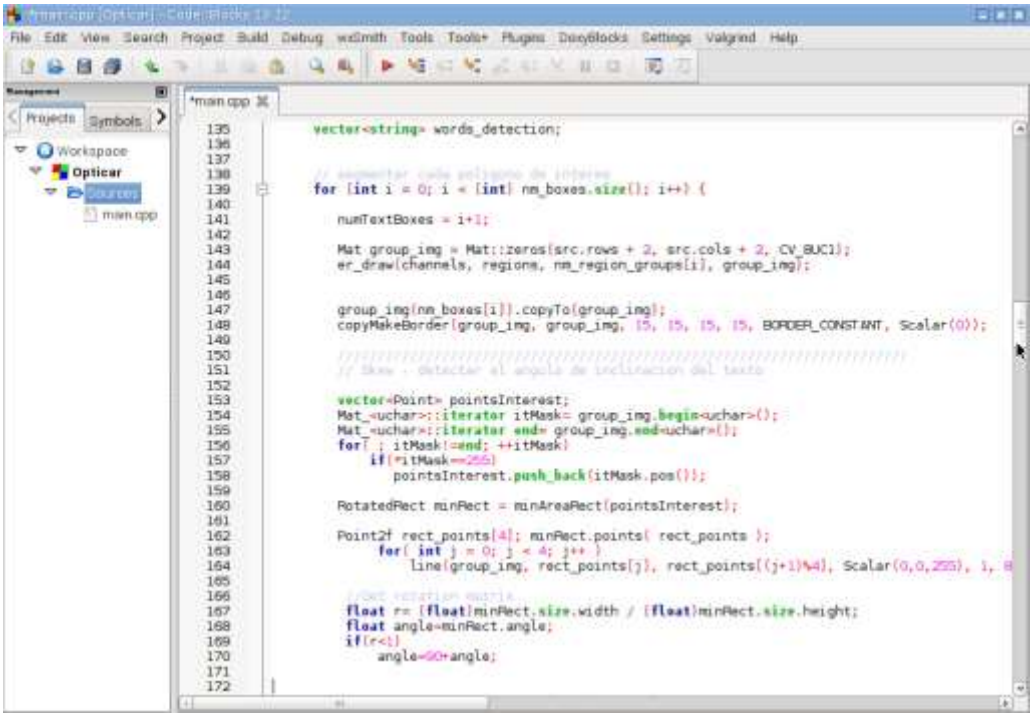

Figura 5: Code::Blocks con proyecto OpenCV

Este proceso no difiere en exceso entre distintos entornos de desarrollo, pues únicamente se ha de indicar al compilador y al *linker* la localización de la instalación de OpenCV, así como de otras dependencias, si las hubiese.

Cabe destacar que el compilador y el *linker* son programas diferentes. Ambos forman parte de la *suite* de compiladores de GNU, GCC. El responsable de compilar condigo C++ es g++, que es un *driver*, encargado de ejecutar el preprocesador (un programa llamado *cpp*) para luego invocar el propio compilador, *cc1plus*, y, finalmente el *linker*, llamado *ld*. El proceso de configuración necesario es el siguiente:

- Ejecutar `pkg-config --libs --cflags opencv` para obtener las rutas de instalación de OpenCV. Aquellas marcadas con –I indican las localizaciones de *includes,* necesarias para configurar el compilador y el indexador usado por el IDE.

- A continuación se procede a configurar las rutas de *includes* (o definiciones), indicando su localización, que, en la mayoría de los casos es `/usr/local/include/opencv`, `/usr/local/include/opencv2` y `/usr/include`. Nótese la existencia de la carpeta `opencv` y `opencv2`.

Esto se debe a que, actualmente la API de OpenCV soporta de manera legada la versión anterior, por lo que debido a cuestiones de compatibilidad se ofrecen las dos opciones.

- El siguiente paso consiste en enlazar con las librerías binarias de OpenCV. Este proceso se realiza indicando las librerías dinámicas a usar, como `/usr/local/lib/libopencv_core.so`, por ejemplo y `/usr/local/lib/libopencv_*.so` para el resto.

- Finalmente, se indican los *flags* (u opciones) específicos al compilador. Dado que Debian 'Jessie' ofrece GCC en su versión 4.9, este soporta múltiples arquitecturas u opciones de compilación específicas, capaces de aprovechar al máximo las capacidades del hardware subyacente. En este caso, las opciones extra son tres: 1) `-O3` - generación de código con máxima optimización 2) `-mcpu=cortex-a7` - optimización de código para la arquitectura Cortex-A7 3) `-mfpu=neon-vfpv4` – optimización que habilita el uso de capacidades SIMD y el uso del coprocesador de cálculo de números de coma flotante.

Periféricos de Raspberry Pi

La placa procesadora principal del vehículo dispone de tres periféricos, sin contar con el microcontrolador Arduino que no se considera como periférico a pesar de estar conectado por USB, pues este ejecuta su propio *software* y posee una alimentación separada. Los periféricos son módulo WiFi, cámara CSI y cámara USB.

1. El módulo WiFi es una unidad Edimax EW-7811Un con chipset Realtek RTL8188CUS.

Figura 6: Edimax EW-7811UN

Las especificaciones técnicas indican la complicidad con los estándares IEEE 802.11 b/g/n y velocidades de transmisión de hasta 150 Mbps en la banda de

2.4 GHz, así como soporte de WPA, WPA2 y WPS. Se trata de un componente muy compacto y de consumo energético reducido a costa de velocidades de transmisión moderadas y radio de funcionamiento limitado. Lo destacable de esta interfaz de red es su popularidad en la comunidad de usuarios de Raspberry Pi al ser una de las primeras tarjetas WiFi USB compatibles OOB (*out of box*), es decir, sin necesidad de instalación de drivers o de realización de configuraciones de algún tipo con la primera generación de las placas Raspberry Pi.

2. El segundo periférico usado es un módulo cámara con interfaz CSI-2 (o Camera Serial Interface).

Figura 7: Módulo cámara de Raspberry Pi

Se trata de una unidad fabricada exclusivamente para la Raspberry Pi. Su aplicación en este proyecto es la captura y posterior envió de video en tiempo real. En términos de diseño, la decisión de usar esta solución viene dada por una serie de factores tales como:

- Tamaño muy reducido – 25 mm x 20 mm x 9 mm y 3 g de peso

- El uso de la interfaz CSI-2 es implementado como un socket ZIF 15 y ofrece una conexión serie directa entre el módulo de la cámara y la CPU de la Raspberry Pi. Al ser una conexión dedicada, el ancho de banda existente es de unos 800 Mbps por canal, siendo 1 Gbps el límite teórico. La implementación usada por este módulo hace uso de dos canales ofreciendo así un ancho de banda total aproximado de 2 Gbps que permite trabajar con video con resoluciones de hasta 1920 x 1080 pixeles y 30 *frames* por segundo.

- El sensor CMOS OmniVision OV5647 ofrece una resolución máxima de 5 megapíxeles (QSXGA) y ofrece distintos modos de transmisión de video: desde QSXGA (2592 x 1944) a 15 fps hasta QVGA (320 x 240) a

120 fps usando el formato RAW H.264 acelerado por hardware, usando el DSP de VideoCore IV de la Raspberry Pi.

- Disponibilidad de software que forma parte de la distribución Raspbian capaz de manejar el *streaming* de video, entre otras características (como imágenes o cambios de ajustes de cámara, tales como formatos de imagen, efectos, modos de exposición u otros), así como disponibilidad de APIs para desarrollo de programas capaces de interactuar con el módulo de cámara.

3. El tercer periférico de la Raspberry Pi es otra cámara usada para el reconocimiento de matrículas y conectada mediante USB.

Figura 8: Creative Live! Cam Sync HD

Se trata de una unidad Creative Live! Cam Sync HD con un sensor CMOS que ofrece una resolución máxima de 1280 x 720 píxeles con capacidad de transmitir video a esta resolución a 30 fps. Las características más destacables que han motivado la elección de este periférico son la compatibilidad OOB (no se requiere cargar ningún modulo del kernel) con la Raspberry Pi, total soporte de V4L2 y bajo consumo. Las dos últimas características son especialmente importantes.

- En primer caso, la compatibilidad con V4L2, que es una API y *driver* que se ejecutan en modo *kernel* del SO, permitiendo a las aplicaciones acceder a las capacidades de la cámara usando una API estándar. En este proyecto V4L2 es usado por OpenCV para acceder a la cámara y obtener frames de la misma. Todo dispositivo compatible con el modo V4L2 aparece como `/dev/video0` al ejecutar el comando `v4l2-ctl --list-devices`, que además, cambiando los argumentos por `--c <option>=<value>` permite el comportamiento del dispositivo o ver los modos de funcionamiento soportados.

- La segunda característica es relevante debido a que los puertos USB de la Raspberry Pi están limitados en términos de corriente máxima

suministrada a los periféricos conectados. En ocasiones esta limitación requiere usar un concentrador (o *hub*) USB con alimentación externa para suplir la falta de corriente. Esto es necesario en caso de conectar periféricos con una alta demanda energética, como puede ser una webcam. En caso de este periférico, el uso de esta técnica no es necesaria al disponer éste de un consumo reducido.

Arduino

La placa Arduino se conecta a la Raspberry Pi mediante una conexión USB, la cual provee una comunicación serie *full-duplex* vía UART (también llamado TTL Serial). Este tipo de conexión permite transmitir un bit tras otro a una velocidad predefinida que oscila entre 9600 bps y 115200 bps. A pesar de que tanto Raspberry Pi como Arduino tienen pines dedicados a comunicaciones TTL, estos no son compatibles entre sí, debido a que la primera hace uso de TTL 3v3 y la última de TTL 5 v. La conexión USB, sin embargo si permite la conexión directa, al implementar TTL 5 v. En la placa Arduino la capacidad de hacer uso de TTL a través de USB es ofrecida gracias al chip FTDI, que es un hardware específico diseñado para convertir transmisiones serie TTL a señales USB, pues la CPU de Arduino no es capaz de realizar tales operaciones.

Periféricos Arduino

Los componentes conectados a la placa Arduino son dos:

1. Puente H dual, basado en el chip L298N. El modelo usado es mostrado en la imagen siguiente:

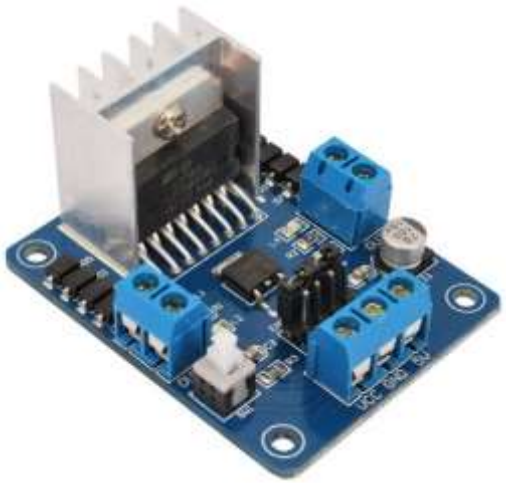

Figura 9: Puente H dual basado en el chip L298N

Este componente es un convertidor de potencia que permite girar un motor de corriente continua en ambos sentidos. Su inclusión es necesaria debido a que

Arduino es capaz de proporcionar únicamente 5 V y 40 mA por pin, valores insuficientes para mover un motor. Por tanto el puente H se encarga de controlar la tensión y corriente muy superiores a los que pueda ofrecer el microcontrolador por medio de una señal enviada por este. De esta forma se puede controlar el motor de una forma precisa por medio de señales PWM (*Pulse Width Modulation* o Modulación por Ancho de Pulsos). El control del puente H es llevado a cabo por tres pines de la placa Arduino. El estado del primer pin (nivel alto o bajo) indica al circuito la dirección del giro del motor, el valor del segundo pin (0 - 255), indica el ciclo del trabajo de la señal PWM, controlando la velocidad de giro del mismo y el estado del tercer pin activa o desactiva el freno. Por último, lo destacable de este circuito son las capacidades eléctricas ofrecidas por el chip L298N: rango de voltaje de entrada de entre aprox. 5 V ~ 35 V, corriente máxima admitida de 2 A por canal y una potencia máxima de 25 W.

2. El segundo componente usado es un servomotor encargado de la dirección del vehículo. Se trata de un Servo analógico OEM de tamaño estándar, bajo coste y que posee características básicas que cumplen con su cometido.

Figura 10: Servo estándar

El conjunto de estas características vienen dadas por el hecho de que esta unidad posee un motor de baja potencia y los engranajes de nylon, lo que limita el esfuerzo de torsión ofrecido. Por otro lado los valores de precisión, velocidad y el centrado cumplen con los mínimos necesarios para las necesidades de este proyecto. En términos de conexiones, este componente se conecta a un pin PWM de Arduino, que se encarga de establecer el ángulo de giro del mismo. La alimentación es provista por una batería auxiliar dedicada, capaz de proporcionar hasta 5 V y hasta 1 A de corriente.

Alimentación

Una parte importante del diseño hardware es el hecho de asegurarse de la correcta alimentación de todos los componentes. Este factor es el responsable en gran parte del correcto funcionamiento de todo el sistema. Un mal diseño lleva a una alimentación deficiente en distintos modos del trabajo de la plataforma y puede causar bajadas de tensión en ciertos componentes críticos, como son los circuitos encargados de procesar los datos o en otro caso, en motores o servomecanismos. Ambos casos son graves, debido a que el sistema es controlado remotamente. Si en algún momento llega a presentarse un problema de falta de corriente, el inconveniente de perder el control de la plataforma.

Para evitar esto, se recurre a múltiples fuentes de energía, encargadas de controlar distintos sistemas por separado. En concreto se utilizan tres fuentes de alimentación, encargadas de proveer de corriente a los componentes que más necesitan. Físicamente las tres fuentes están implementadas por dos baterías de ion-litio de gran capacidad y con finalidades distintas.

- La primera batería es una unidad de 340 gramos de peso, 4500 mAh de capacidad, un voltaje de salida que varía entre 12.6 V y 10.8 V y capaz de proporcionar 1.5 A de corriente continua.

Figura 11: Batería principal de alta capacidad

Esta batería es la encargada de alimentar al motor principal del vehículo al tratarse de un motor eléctrico con escobillas de 30 mm, este puede consumir más de 1 A a 10 V de forma continua y aún más en bruscas aceleraciones. Dado que esta fuente puede cumplir con las exigencias de alimentar el motor, la corriente sobrante (>300 mA) se utiliza para alimentar el puente H y haciendo uso del regulador de voltaje de éste, es capaz de reducir el voltaje entrante a 5 V y alimentar a la placa Arduino (aunque la reducción de voltaje no es estrictamente necesaria, al llevar Arduino su propio regulador).

Aunque si esta batería fuese la única fuente de alimentación disponible, las bajadas de tensión que ocurren durante los intervalos de tiempo, cuando la demanda de energía es más elevada, provocarían que el resto de componentes alimentados se quedasen sin la alimentación adecuada con el correspondiente peligro de perder el control sobre el sistema. Por este motivo la parte lógica cuenta con una fuente separada. Debido al inconveniente que el componente lógico más importante, la Raspberry Pi, por diseño acepta únicamente 5 V, la fuente debe de ser capaz de proporcionar unos 2 A a 5 V. Esto se debe al consumo máximo combinado de la Raspberry Pi y sus periféricos (800 mA de la Raspberry Pi, 200 mA del módulo de la cámara y 1 A consumido entre el módulo WiFi y la Webcam USB).

- Para cumplir con estos requisitos se utiliza una segunda unidad de 240 gramos, 10400 mAh de capacidad y un voltaje de salida de 5 V.

Figura 12: Batería secundaria con salida doble

Lo destacable de esta batería es la disponibilidad de dos puertos USB a través de los cuales es capaz de proporcionar una corriente de 1 A y 2 A, respectivamente. Por ello, el puerto que genera 2 A se utiliza para alimentar a la Raspberry Pi y sus periféricos. El puerto restante se utiliza para alimentar al último componente energéticamente relevante, el servomotor. Dado que este requiere unos 300 mA y 5 V, la salida USB secundaria es ideal para alimentarlo por separado sin tener que recurrir a un regulador de voltaje adicional.

Al garantizar la disponibilidad de corriente para cada uno de los componentes en cualquier circunstancia, este diseño resulta óptimo para proveer de energía un sistema como el propuesto.

Chasis

Por ultimo cabe señalar que por muy sofisticada que sea la lógica que controla la plataforma, las capacidades de esta están sujetas a las limitaciones de la plataforma portante, al tratarse de un sistema móvil. Para mantener un nivel operativo mínimo, se

ha recurrido a un chasis de coche de radiocontrol dotado de tracción integral permanente, suspensiones y dirección controlada por servomotor.

Figura 13: Plataforma móvil usada en el proyecto

El hecho de contar con estas características garantiza, al menos en cierta medida, la posibilidad de poder ser conducido de una forma óptima. Esto implica una mejor experiencia de usuario y por tanto, una mayor eficacia.

Software

La parte software, encargada de controlar el comportamiento del vehículo, así como la de permitir la interacción del usuario con el mismo se compone de dos partes:

1. La parte que se ejecuta fuera del vehículo (por el propio usuario), formado por el programa de escritorio, encargado de mandar las señales de control del mando y visualizar el video en tiempo real recibido, así como por el dispositivo Android, que manda las señales de control obtenidos a partir de las entradas táctiles y basadas en movimiento.

2. La que se ejecuta en la circuitería montada en el mismo (embebida). Esta parte coordina el *hardware* en función de las señales recibidas y manda el video capturado al *software* de usuario.

A continuación se describe el *software* de usuario, seguido de la parte embebida.

Software de escritorio

En primer lugar se describe la parte del software de escritorio, usado por el operador para manipular el vehículo con el mando y poder ver el video transmitido desde el vehículo. Esta aplicación está hecha en C++ usando Qt framework. Los componentes que forman el programa son los siguientes:

- Instancia del mando conectado provista por la librería *XInput*
- Hilo dedicado al *polling* del estado del mando que da valores al *input* del mismo
- Hilo dedicado al envió de los valores del input del mando al vehículo
- Hilo dedicado a la escucha de mensajes provenientes del vehículo
- Hilo principal del programa, encargado de actualizar la GUI
- Campo de texto reservado para la IP del vehículo y el botón encargado de arrancar los hilos cuando se haya notificado la IP del software de usuario al vehículo

El funcionamiento del programa se puede describir de la siguiente forma:

1. En primer lugar arranca *netcat*, que redirigirá la entrada del *stream* de video que se espera recibir en un puerto concreto vía *pipe* a MPlayer, que a su vez se utiliza para el visionado del video recibido. Concretamente, el comando a ejecutar es el siguiente: `“nc64 -L -p 5001 | mplayer -fps 31 -cache 512 –“` Esta configuración por parte del cliente permite una reproducción del video muy eficiente, pues se trata del visionado de *frames* de video H.264 que fueron enviados empaquetados en tramas UDP en un reproductor que fue configurado para no usar cache para *buffering* (pues esto implica aparición de latencias).

2. En segundo lugar arranca el hilo principal, carga la interfaz gráfica y conecta los *signals* con *slots* (mecanismo de paso de datos en Qt), usado aquí para intercambiar los datos entre los hilos y sincronizarlos con la GUI.

3. En tercer lugar se procede a obtener una instancia del mando, siempre y cuando éste se encuentre conectado. Este objeto permite obtener acceso a valores de controles analógicos y estado de los controles digitales.

4. A continuación arranca el hilo encargado de leer los valores del mando, aplicar una función de reducción (también llamada zona muerta) necesaria debido a la alta precisión de joysticks analógicos. Sin esta característica obtener la posición 0 o “centrado” sería imposible (debido a 16 bits de resolución de cada uno de ellos). Finalmente se recurre a una función de conversión lineal, que convierte los valores nativos ofrecidos por la API del mando (de 0 a 1) en aquellos que serán usados por el servo y el motor.

5. Finalmente se forma el JSON que tiene el siguiente formato: `"[{"offset": 0,"steering": 88,"throttle": 257}]"` y que luego será mandado al vehículo.

El entorno de desarrollo usado con el proyecto es mostrado en la imagen siguiente:

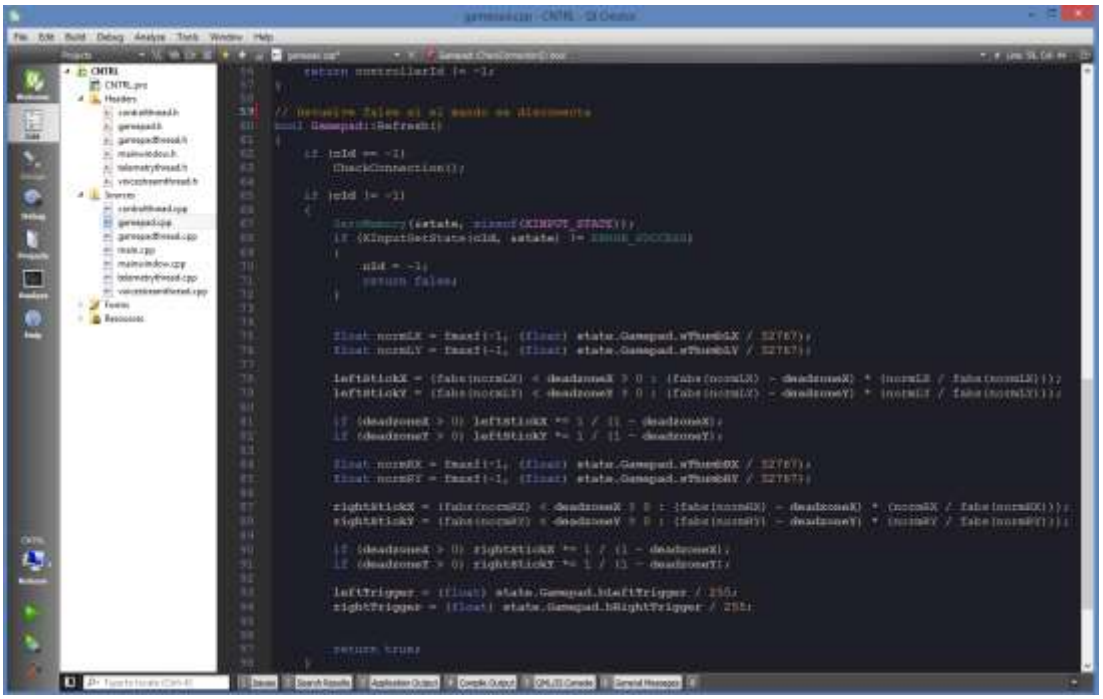

Figura 14: Qt Creator con el proyecto CNTRL de escritorio

Las partes software más interesantes usados en el proyecto son las siguientes:

- La inclusión de *LIBS += -lXInput* en la configuración del proyecto y de la cabecera *#include <XInput.h>* permite obtener acceso al estado del mando usando: `XINPUT_GAMEPAD *Gamepad::GetState() { return &state.Gamepad; }`

- La comunicación entre los hilos se realiza usando el mecanismo estándar de Qt llamado *signals* y *slots*. El uso es tal como sigue:

`QObject::connect (gmpdThread, SIGNAL (leftStickXvalue(float)), cntrlThread, SLOT (setGamepadLeftStick(float)));` En este ejemplo, en caso de producirse un evento (como la actualización del valor), el hilo del *polling* del mando (representado por la instancia *gmpdThread*) notificará al hilo responsable de enviar señales de control vía UDP (*cntrlThread*) el valor *float* del joystick X del mando. Este mecanismo es muy eficiente, compacto y no requiere sincronizaciones al ser *thread-safe*.

- En Qt un objeto JSON es formado usando un tipo de dato especial, llamado `QVariant`, que actúa como unión para los tipos de datos comunes. Por ejemplo para enviar un JSON con tres valores, como es el caso de este proyecto, primero estos se asignan a un `QVariantMap`, luego se convierten a `QVariant` y, finalmente, el `QJsonDocument` se obtiene a través de `QJsonDocument::fromVariant(data)`

El siguiente esquema DRAKON [1] ilustra el funcionamiento del programa:

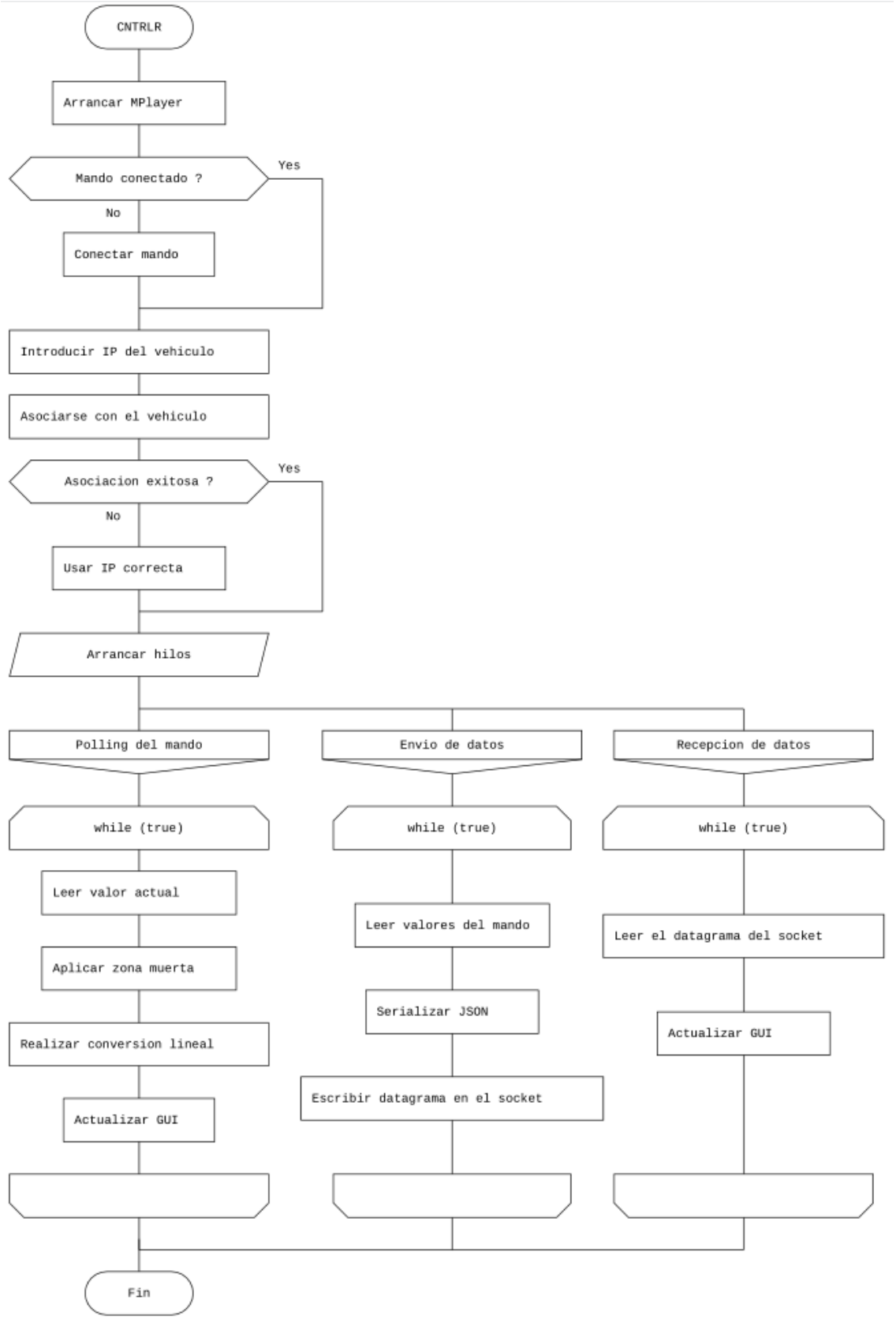

Figura 15: Esquema DRAKON que ilustra el funcionamiento del software de escritorio

La interfaz gráfica del usuario muestra varias ventanas, separando la ventana de video de la de controles, permitiendo así al usuario elegir su disposición y tamaño de forma independiente. En la siguiente imagen se puede ver el detalle de la GUI, la consola de *debug* y de la prueba de latencia de transmisión de vídeo.

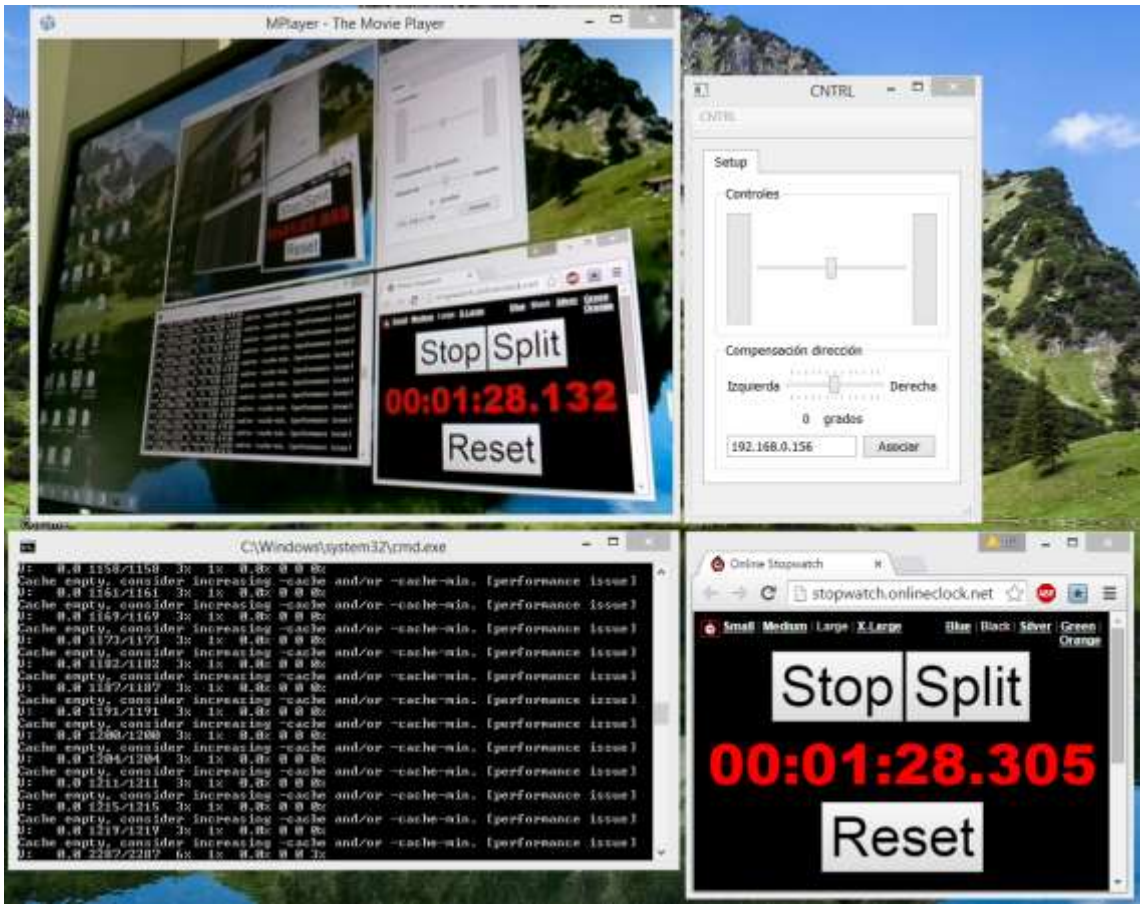

Figura 16: Detalle de la GUI de software de escritorio

La ventana principal de la aplicación muestra dos barras verticales y una horizontal. Estos elementos muestran el estado en el que se encuentra el mando. Las barras verticales corresponden a acelerador y el freno, respectivamente, mientras que la barra horizontal corresponde a la dirección. El panel llamado “Compensación de dirección” sirve para ajustar de forma fina la dirección del vehículo en caso de desvío hacia izquierda o derecha. Pulsando los botones L o R del mando una o varias veces, la dirección se centra en el valor deseado (+1, +2 o más para la derecha y -1, -2 o menos para la izquierda). De esta forma se puede compensar el centrado incorrecto de la dirección del vehículo dinámicamente en cualquier momento.

Software Android

La aplicación que se ejecuta en un dispositivo Android tiene como finalidad ofrecer una serie de controles alternativos tomando el *input* de la manipulación del propio dispositivo por parte del usuario. Para lograrlo, se sigue el mismo principio que en la aplicación de escritorio, con la única diferencia que los valores de entrada leídos no provienen del mando, sino del acelerómetro o de la posición del dedo en la pantalla. Aparte de este detalle, el funcionamiento de la aplicación es muy similar: se procede a arrancar un hilo que obtiene los datos de uno de los dos tipos de entrada para luego serializarlo en un JSON con el mismo formato que la aplicación de escritorio (`"[{"offset": 0,"steering": 88,"throttle": 257}]"`) y a continuación enviarlo al vehículo. Las interfaces de la aplicación móvil son las siguientes:

Figura 17: Pantalla principal de la aplicación móvil

La pantalla principal permite elegir entre el tipo de control deseado.

Figura 18: Pantalla del control basado en movimiento muestra los datos en tiempo real

Por último, el control táctil muestra al usuario el área de entrada y los valores obtenidos.

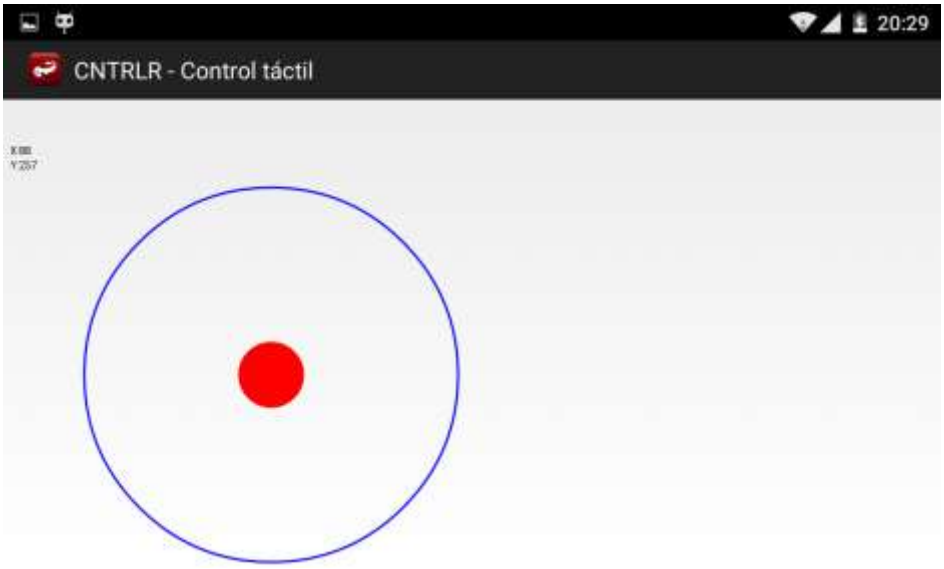

Figura 19: Pantalla de control táctil con el canvas y los valores leídos

Al igual que en la aplicación de escritorio, los valores leídos de distintas fuentes de entrada no se corresponden con valores deseados, por lo que se convierten mediante una función de conversión lineal de un rango de valores a otro. Por ejemplo, en caso de los valores leídos del acelerómetro, estos van desde -255 a 255 para ambos ejes, mientras que en caso del control táctil los valores se tienen que calcular conforme la posición del toque dentro del círculo respecto al ancho y alto de la pantalla.

Software embebido

La parte del software embebido corresponde a una serie de programas que se ejecutan en el propio vehículo. Todos se ejecutan en la Raspberry Pi, excepto uno que es ejecutado por la placa Arduino.

Software Raspberry Pi

Raspberry Pi realiza las siguientes operaciones: 1) recepción de comandos de control y su reenvió a Arduino 2) reconocimiento de matrículas y el envío de resultados al sistema de escritorio y 3) envío del *stream* de video al sistema de escritorio. A continuación estas operaciones se analizan en detalle y se asocian al software que los lleva a cabo.

1. La recepción de los comandos de control y su reenvío al puerto serie correspondiente a Arduino se lleva a cabo por un script Python que realiza las siguientes operaciones:

   - Crear una instancia del *socket* UDP escuchando en el puerto deseado
   - Crear una instancia del puerto serie en el dispositivo '/dev/ttyUSB0'
   - Ejecutar un bucle infinito
   - *Parsear* el contenido de cada datagrama recibido como un JSON
   - Obtener los elementos del JSON (*steering*, *throttle*, *offset*)
   - Formar el *String* con estos valores en el formato aceptado por Arduino
   - Escribir el *String* formado en la instancia del puerto serie
   - Volver a ejecutar para la próxima iteración del bucle

2. El reconocimiento de matrículas se lleva a cabo por la aplicación Opticar, compilada nativamente y cuyo funcionamiento se describe en detalle en el apartado correspondiente. Para el envío de los resultados, Opticar establece un *pipe* a otro script Python que lee los resultados desde la salida estándar y los envía al sistema de escritorio escribiéndolos sobre la instancia de un *socket* (con dirección IP el puerto del sistema de escritorio como destino) en cada iteración de un bucle infinito.

3. Por último, el envío de video en tiempo real se realiza redirigiendo la salida del programa *raspivid* a netcat con la dirección IP y el puerto del sistema de escritorio. Concretamente, el comando usado es el siguiente: "`raspivid -t 999999 -o - -n -w 640 -h 480 -fps 20 | nc %IP_DESTINO% 5001`". Este método permite el envío de video de una manera muy eficiente debido a dos motivos:

   - El envío del *stream* de video a través de netcat elimina el *overhead* existente en protocolos de *streaming* tales como RTP o RTSP al enviarse los datos empaquetados en una trama UDP. Como punto negativo, no hay muchos reproductores de video capaces de reproducir video *raw* desde la salida estándar.

   - raspivid es un programa que forma parte de la distribución Raspbian y ha sido diseñado para funcionar exclusivamente con las Raspberry Pi y sus módulos cámara dedicados. Debido a esto, es capaz de aprovechar las API de bajo nivel OpenMAX IL para sacar partido al hardware de procesador multimedia de la Raspberry Pi, VideoCore 4, que codifica el stream de video de la cámara al formato H.264 por hardware de una forma muy eficiente. Esta operación es posible en las dos generaciones de la placa, al compartir ambas la misma GPU. La siguiente imagen indica la arquitectura software de acceso al hardware multimedia:

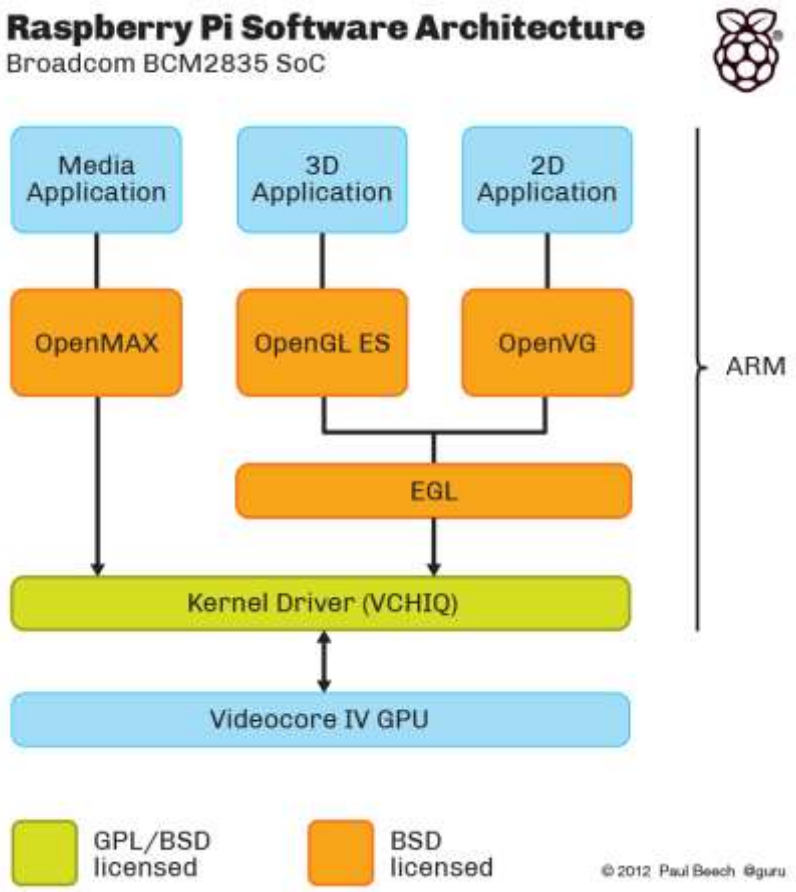

Figura 20: VideoCore 4 APIs

Software Arduino

La placa Arduino realiza las siguientes operaciones:

1. Abrir la conexión serie con la velocidad adecuada (la misma que la Raspberry Pi), de lo contrario los datos recibidos serán corruptos e ilegibles.

2. Declarar e inicializar los pines a usar (3 pines para controlar el motor DC usando el puente H y un pin para controlar el *Servo*). Todos los pines son *OUTPUT*

3. Entrar en el bucle principal del programa

4. En cada iteración del bucle esperar al *String* de comandos a través del puerto serie.

5. En caso de recibir el String entrante y si éste este bien formado (%valor%s,%valor%t, *s* y *t* vienen de *steering* y *throttle*), obtener los valores de las variables recién llegadas.

6. Para el valor del motor (*throttle*): establecer la dirección de giro (hacia delante o atrás), quitar el freno, escribir el valor de la velocidad deseada en el pin correspondiente.

7. Para el valor de la dirección (*steering*): girar el servo a la posición adecuada, indicada por el valor de la variable.

8. *Resetear* el *String* recibido.

9. Repetir el proceso con la próxima iteración del bucle.

Reconocimiento de matrículas (ANPR)

En este apartado se describe el diseño e implementación del proyecto de reconocimiento automatizado de matrículas de automóvil usando dispositivos móviles o embebidos. La herramienta usada es la librería OpenCV que posee una licencia BSD. Para el diseño inicial se utilizó un proyecto Visual Studio 2013 enlazado con las librerías dinámicas de OpenCV. Este entorno de trabajo demostró ser más propicio para un desarrollo y prototipado rápidos al no tener restricciones como el que sí pueden llegar a tener un desarrollo para un dispositivo móvil o de prestaciones reducidas. Dicho esto, la primera implementación del sistema lo constituye una aplicación de escritorio. Posteriormente, se realiza la portabilidad a un sistema Raspberry Pi y Android manteniendo el mismo código, responsable de los procesos de tratamiento de imagen, como ejecutable en caso de la Raspberry Pi y como librería dinámica nativa en caso de Android.

Dada la complejidad del problema abordado, la solución no podía ser única. De hecho existe una multitud de posibles soluciones. La diferencia entre las mismas las pueden dictar las métricas del funcionamiento del producto final. A continuación se describen pasos básicos del funcionamiento del programa propuesto por [2] (aunque se comparten ciertos pasos que sí son comunes a otras posibles soluciones) y además cada uno de ellos será descrito en detalle.

Obtención de la foto del vehículo

Ésta se puede obtener de una imagen estática o extraer de un *frame* del *preview* de una cámara. El primer caso es muy directo y es usado en la Raspberry pi, pues en este caso es la solución habitual, mientras el segundo caso, enfocado a dispositivos Android, permite elegir entre usar la implementación de una cámara del sistema con el que se trabaja (android.hardware.Camera) u optar por una clase envoltorio de la misma provista por OpenCV (org.opencv.android.JavaCameraView). Para esta aplicación se optó por la última, dado que esta solución permite obtener las imágenes en una estructura de datos Mat, que es usada por OpenCV a lo largo de todo el proceso.

Primera solución

Segmentación de la imagen para obtención de la matricula

Una vez obtenida la imagen inicial, se procede a extraer el polígono de interés, que, idealmente, es la matricula sin los elementos que la rodean (marcos, símbolos) pues estos pueden dificultar la distinción entre caracteres que forman la misma y ruido existente (como, por ejemplo, sombras, suciedad, imperfecciones, entre otros). Dado que este paso implica muchas operaciones, a continuación solo se describirán las más importantes. Este paso sí admite distintos diseños, tales como: clasificación usando SVM (*Support Vector Machine*), extracción de características usando cascadas de Haar, reconocimiento de patrones y reconocimiento de contornos. Debido a la relativa simplicidad y rapidez de diseño e implementación en la primera versión del programa se optó por la última opción.

La idea consiste en detectar un polígono cerrado de color uniforme que cumpla ciertas características (relación de aspecto 52cm / 11cm, color de fondo uniforme, determinados tamaños máximos y mínimos y una alta densidad de líneas verticales).

Las operaciones son como siguen:

- Conversión a escala de grises y suavizado
- Filtro Sobel
- Operación de umbral adaptable
- Operación morfológica de cierre
- Detección de contornos

Gráficamente, estas operaciones se representan de la siguiente forma y orden:

Figura 21: Filtro Sobel

El filtro Sobel se utiliza para hallar los bordes verticales y la primera derivada horizontal. La derivada es la función matemática que en OpenCV se define como:

```
void Sobel(InputArray src, OutputArray dst, int ddepth, int xorder, int yorder, int ksize=3, double scale=1, double delta=0, int borderType=BORDER_DEFAULT);
```

Figura 22: Umbral adaptable

Mediante el uso del umbral adaptable se obtiene una imagen binaria con un valor de umbral obtenido a través del método Otsu. Este algoritmo requiere una imagen de 8 bits como entrada y devuelve el valor de umbral como resultado.

Su uso es el siguiente:

```
Mat img_threshold;
Threshold (img_sobel, img_threshold, 0, 255, CV_THRESH_OTSU + CV_THRESH_BINARY);
```

Finalmente, aplicando la operación morfológica de cierre, se puede eliminar los espacios vacíos entre cada borde vertical y conectar los regiones que contienen un número elevado de bordes verticales. En este paso se puede obtener las regiones que pueden contener una matrícula.

Figura 23: Operación morfológica de cierre

Figura 24: Detección de contornos

Una vez finalizado el paso de detección de contornos se obtienen candidatos de contener la matrícula. En la imagen de arriba se puede observar que de los tres contornos detectados, dos son erróneos. Para descartarlos, todos los candidatos se recortan y se tratan por separado. Los contornos obtenidos son los siguientes:

Figura 25: Contorno erróneo nº1

Figura 26: Contorno erróneo nº2

Figura 27: Contorno a priori correcto

Reconocimiento de la matricula

Una vez obtenidos los polígonos candidatos se puede proceder directamente al siguiente paso e invocar el programa OCR (Tesseract) para cada uno de los polígonos para éste devuelva el *String* que compone la matrícula esperando un resultado para el polígono correcto y ninguno para polígonos erróneos. Sin embargo esta opción no es acertada, pues el OCR siempre intenta obtener un texto de la imagen y en caso de que ésta no esté bien formada, el texto resultante será completamente erróneo. Incluso después de aplicar la operación de umbral al polígono correcto, el resultado obtenido del Tesseract (OCR) puede que esté por debajo del 50% de caracteres reconocidos. Como se puede observar en la imagen de abajo, la bancarización pone de manifiesto las imperfecciones existentes en la imagen. Este hecho trae la necesidad de reconocer los caracteres que forman la matrícula y reconstruir la imagen basándose en los mismos. Para ello se toma como base un rectángulo completamente negro que actuará de fondo. A continuación se procede a detectar contornos en el propio polígono. Una vez finalizado el proceso para todos los polígonos se comparan los resultados, se filtran los contornos que incumplen la condición de ser posibles caracteres (por relación de aspecto, tamaños, áreas) y tras comparar el número de contornos correctos para cada polígono se desechan los que incumplan esas condiciones. Finalmente se procede a unir el fondo base con los caracteres detectados, previamente aplicándoles ciertos filtros, quedando la imagen binarizada y con poco ruido y deformaciones.

Figura 28: Contorno binarizado mediante una operación de umbral

Figura 29: Caracteres subsegmentados

Figura 30: Caracteres subsegmentados combinados con un fondo negro

Reconocimiento óptico de caracteres

El paso final es el de pasar la imagen resultante al programa de reconocimiento óptico de caracteres que devolverá el *String* deseado. Para dicho fin se utiliza Tesseract de Google, que posee una licencia Apache. Además para un reconocimiento óptimo se utiliza un archivo de entrenamiento reducido. El uso y las limitaciones de Tesseract se comentan más en detallen en apartados siguientes.

Conclusiones y análisis de esta posible solución

En estos momento el programa es estable y funcional con una velocidad de detección muy alta (<1s y >80\%) en condiciones normales. Estas condiciones consisten en una matrícula de 52/11cm, buena iluminación y a una distancia de entre 1.5 y 3 m. Antes de proceder con futuros desarrollos se requiere un estudio de las prestaciones del programa actual en entornos reales. Para ello habría que analizar los fallos cometidos por la aplicación y en que contextos se producen. A continuación se procedería a ajustar los parámetros del diseño existente para intentar compensar la pérdida de rendimiento. Si, una vez puesto a punto el diseño actual, el rendimiento sigue siendo insatisfactorio, entonces se podría valorar el cambio hacia un sistema basado en algún sistema inteligente (SVM, Perceptrón multicapa u otros). Por el momento se quiere prescindir de estas soluciones por ser demasiado costosas de entrenar, al requerir un entrenamiento basado en procesando >2000 fotografías de vehículos.

Segunda solución

A diferencia del trabajo anterior, que sirvió como punto inicial en el desarrollo del software de reconocimiento de matrículas, esta propuesta parte de la idea de búsqueda de polígonos de interés, sustituyéndola por la búsqueda de regiones ER. Este algoritmo permite la detección de regiones que contienen texto en una imagen basándose en características inherentes al texto (elevado número de líneas verticales, agujeros, contraste con el fondo, ente otros) en vez de la detección de contornos, como era el caso de la solución anterior. Este cambio permite la detección de cualquier tipo de texto en una imagen. Aplicándolo a las matrículas, se permite la detección de cualquier tipo de

matrícula nacional o extranjera debido a que el proceso de segmentación se orienta hacia la detección del texto en vez de un posible polígono que puede contenerlo. El diseño del algoritmo utilizado es explicado en el trabajo académico titulado "*Real-Time Scene Text Localization and Recognition*" de Neumann y Matas [4]. Dada su relativa eficacia, la implementación del mismo fue incluida en la librería OpenCV como un módulo externo (modulo texto [3]). Uno de los objetivos de este proyecto tiene como objetivo portar a Raspberry Pi y a Android la lógica del módulo texto de OpenCV, adaptarlo y utilizarlo en un contexto de reconocimiento de matrículas.

Partes

El trabajo con este proyecto obliga a separarlo en tres partes, cada uno de los cuales constituye un pequeño proyecto aparte: lógica Java (en el caso de *port* a Android), lógica C++ y compilación de OpenCV. La última parte es necesaria debido a que el módulo de texto no forma parte de OpenCV ni OpenCV4Android. Esto obliga a compilar todo lo anterior exclusivamente para Android y Raspberry Pi, incluyendo el mencionado módulo de texto. En cuanto a la aplicación Android resultante, ésta se compondrá de una parte Java que se encarga de inicializar las interfaces y de cargar las librerías dinámicas correspondientes a la parte de lógica C++. Básicamente, la parte Java corresponde a una aplicación Android común que

1. Carga e inicializa todo lo necesario para el funcionamiento del programa.

2. Coge *frames* del *feed* de la cámara y las pasa a la subrutina de la librería C++

3. Obtiene el resultado como *String* (si existe), se confirma su veracidad (usando expresiones regulares) y se muestra en la interfaz del usuario.

El siguiente esquema DRAKON ilustra el funcionamiento de la aplicación híbrida Android:

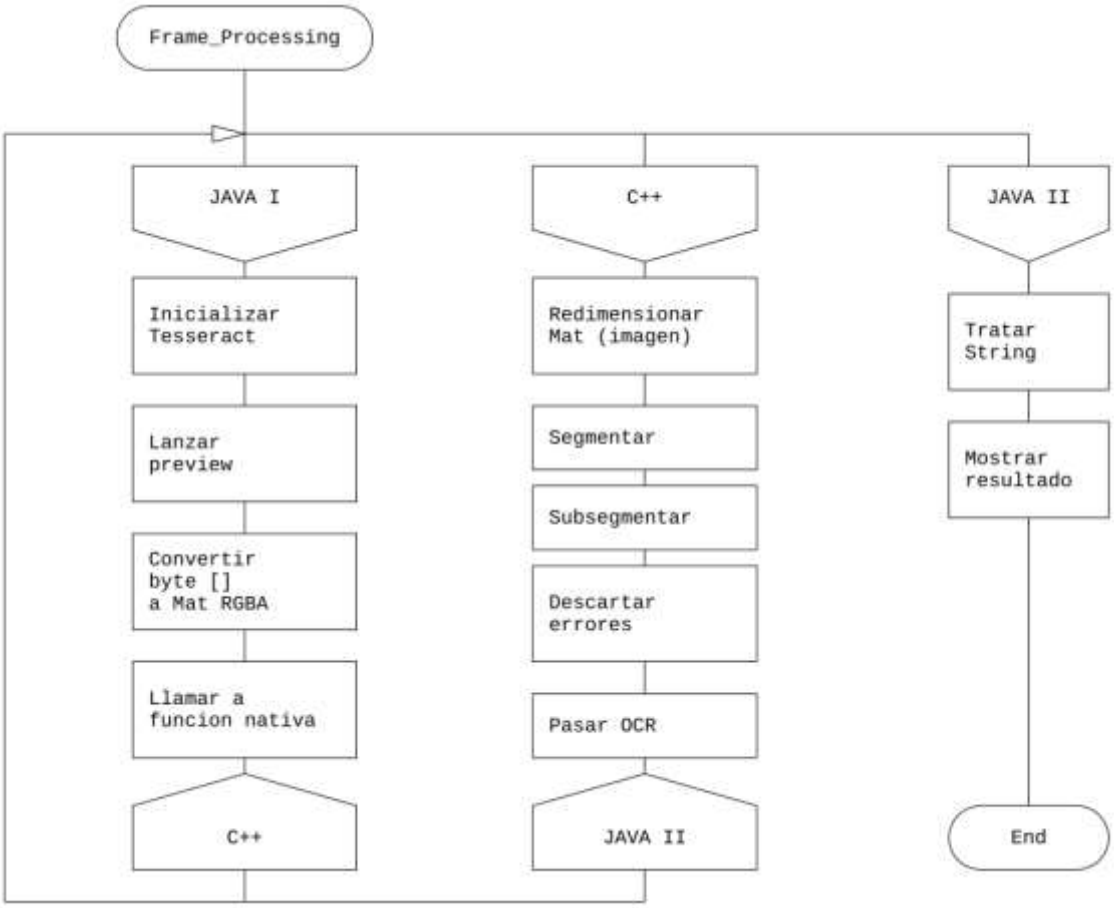

Figura 31: Funcionamiento de la aplicación híbrida Android de reconocimiento de matrículas

Como se puede observar, todo el procesamiento: segmentación y reconocimiento OCR es realizado por la librería dinámica compilada nativamente. En caso de la Raspberry Pi, se prescinde de la parte Java y se usa la lógica desarrollada en C++ directamente, al no estar sujeto a limitaciones del SO debido a que este entorno se comporta como una máquina de escritorio. El enfoque hibrido permite usar completamente las posibilidades que ofrece OpenCV, en vez de estar limitado a su *port* para Android, además de poder añadir código ya existente al proyecto sin necesidad de reescribirlo en Java. El siguiente esquema muestra el funcionamiento de la aplicación híbrida al procesar *frame* del *feed* de la cámara.

El mecanismo que permite a Java interactuar con C++ se llama JNI y el trabajo con él, así como la estructura de la parte nativa de la aplicación se describen en el **Anexo I**.

Inicialización

La parte de inicialización comprende la comprobación de que los archivos necesarios existen. En total son tres archivos: *trained_classifierNM1.xml*, *trained_classifierNM1.xml* y *eng.traineddata*. Los dos primeros son usados por los

clasificadores en el primer paso de la segmentación de la imagen, mientras que el segundo es el archivo de entrenamiento de Tesseract. Si todos los archivos existen en sus rutas correspondientes, se llama a la función nativa que carga los archivos. Si la carga se llevó con éxito, se inicia el *preview* de la cámara con la resolución deseada.

Obtención de la imagen

A diferencia de la propuesta anterior, se ha sustituido la cámara de OpenCV (org.opencv.android.JavaCameraView) por la cámara estándar de Android (android.hardware.Camera). Este cambio se debe a que la cámara de Android permite la captura de *frames* con *buffer*, lo que elimina el *overhead* provocado por la recolección de basura, lo que a su vez resulta en un menor consumo de recursos y, sobretodo, en un *preview* más fluido. Además la lista de resoluciones efectivas (de imagen y de *preview*) viene definida por el SO y no OpenCV, siendo más compatible con dispositivos Android. Como inconveniente cabe destacar la necesidad de conversión del espacio de color NV21 (usado por defecto en todos los dispositivos Android) a RGBA (usado por OpenCV).

Figura 32: Imágen obtenida del preview de la camara

Segmentación

La parte de segmentación consiste en pre y *postprocesamiento* de imágenes tratadas por el módulo de texto de OpenCV. El primer paso es la redimensión de la imagen original a tratar. Esto es necesario porque el proceso de clasificación de regiones ER es computacionalmente muy costoso. Reduciendo el tamaño de la imagen, el tiempo de procesamiento de la imagen se reduce drásticamente. A continuación se clasifican los canales que forman la imagen por separado. Este paso es el más lento de todo el proceso. Para acelerarlo se puede paralelizar el proceso dividiendo las iteraciones del bucle entre múltiples hilos. Las siguientes formas de paralización son disponibles: OpenMP, cv::ParallelLoopBody y *pthreads*.

Una vez obtenido el polígono segmentado, se comprueba su rotación respecto a la horizontal. Si el ángulo supera 8 grados, el polígono se descarta debido a que es poco

probable que el OCR devuelva un resultado satisfactorio. Esto se puede corregir aplicando una transformación afín al polígono y corrigiendo su ángulo.

Figura 33: Segmentación defectuosa

Figura 34: Segmentación corregida

El siguiente paso es *subsegmentar* el polígono para obtener el número de contornos que éste contiene. Si su número es demasiado alto o bajo, entonces es muy poco probable que se trate de texto. Asimismo, se comprueba la altura de cada uno de los contornos respecto a la altura media. Si las alturas difieren, significa que algún contorno puede que no sea una letra o puede que si lo sea, pero debido a una segmentación incorrecta, ésta es defectuosa o mal formada. Si tras este paso no se detecta ninguna anomalía, entonces se procesa con el OCR

OCR

El proceso de reconocimiento óptico de imágenes se lleva a cabo por el programa dedicado a tal fin, llamado Tesseract. Este programa toma como entrada una imagen y hace uso de su lógica y del archivo de entrenamiento para un lenguaje en concreto que contiene las descripciones de las letras que forman el lenguaje a reconocer. En caso de reconocimiento de matrículas, este lenguaje es acotado a números y mayúsculas latinas (prescindiendo de minúsculas y caracteres especiales). Tesseract acepta una serie de configuraciones que permiten ajustar su comportamiento. En primer caso son los conjuntos de caracteres admitidos, el ajuste y la corrección de la posición (o *layout*) del texto en la imagen y el tipo de segmentación de texto elegido (orientado a letra, palabra, línea o página). Para obtener el máximo rendimiento de Tesseract, la imagen de entrada debe ser libre de imperfecciones, en blanco y negro (color y escalas de grises ofrecen un rendimiento muy pobre) y con una resolución mínima de 96dpi. Aun así, Tesseract no ofrece una tasa de reconocimiento excepcionalmente alta, a no ser que este haya sido ajustado para reconocer un tipo de imágenes en concreto y sus opciones hayan sido ajustadas en consecuencia.

Una de las formas más eficaces de aumentar el porcentaje de éxito en reconocimiento óptico de imágenes es mejorar el archivo de entrenamiento o generar uno particular, enfocado al trabajo con unas fuentes específicas. En caso de las matriculas de automóviles, distintos países usan tipos de fuentes distintas en sus matrículas (en caso de las matriculas nacionales, este es DIN 1451 y en caso de las matriculas alemanas es el FE Mittelschrift) Dado que las dos fuentes difieren entre sí, para un correcto reconocimiento se requiere un archivo de entrenamiento generado a partir de muestras de ambas fuentes. Este proceso se puede realizar usando las fuentes sintéticas y posteriormente generando el archivo de entrenamiento, pero el mejor resultado es

obtenido usando los extractos de caracteres que forman las matriculas, generando una imagen de entrenamiento y posteriormente, usándolo. Por tanto para aumentar la precisión del proceso de reconocimiento se requiere realizar un entrenamiento de Tesseract. El proceso es el siguiente:

1. Obtener segmentos de caracteres (normalmente se extraen y se guardan durante la fase de *subsegmentación*) Estos son un ejemplo de carácter de entrenamiento:

Figura 35: Caracteres usados en entrenamiento

2. A continuación se hace uso de un pequeño programa auxiliar (hecho en OpenCV) que recorre los caracteres ordenados por carpetas y los copia en una sola imagen de entrenamiento, tal como la que se indica a continuación:

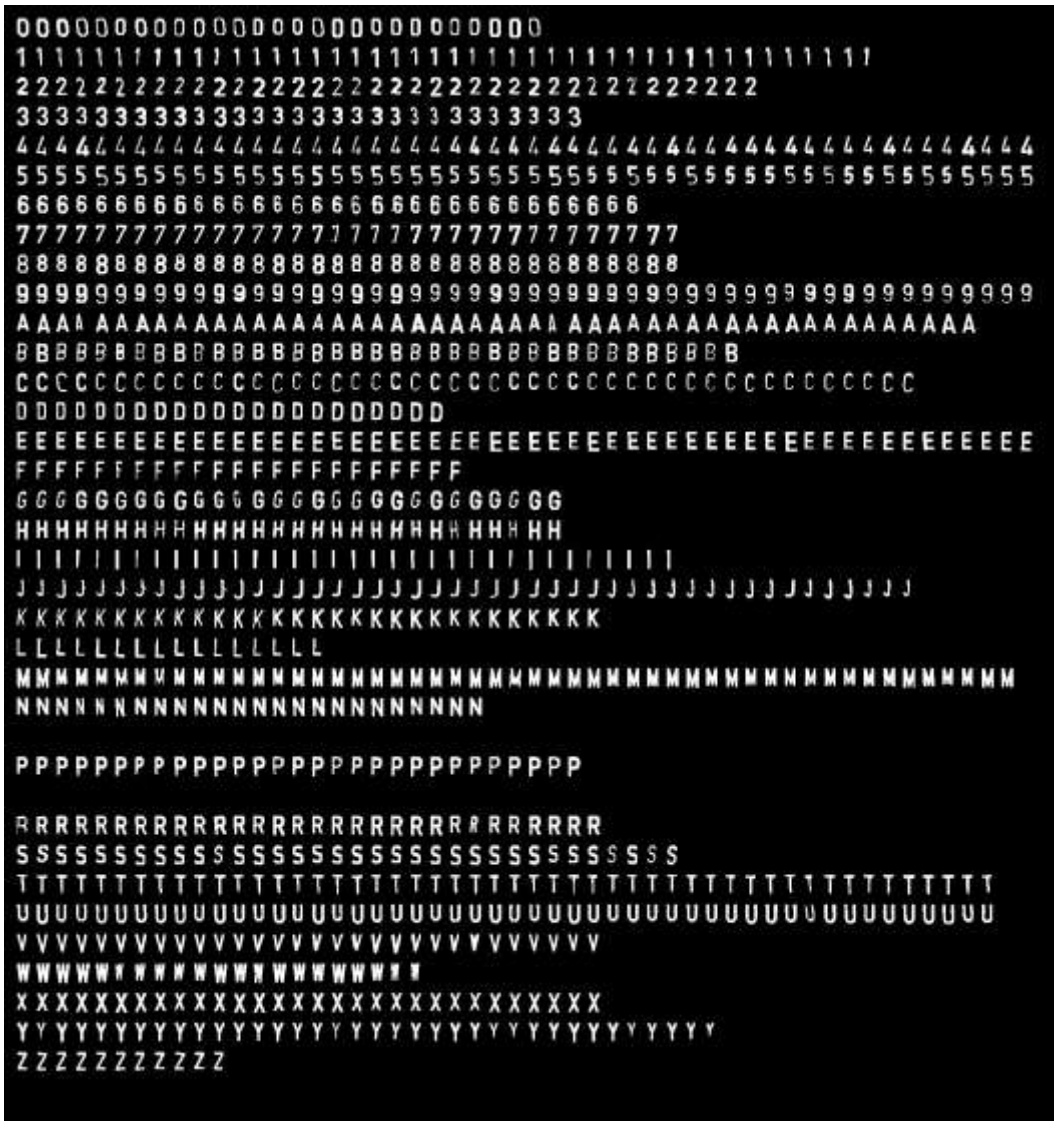

Figura 36: Imagen de entrenamiento de Tesseract

3. continuación se hace uso de distintas herramientas disponibles, cuya finalidad es facilitar el proceso de generación de un nuevo archivo de entrenamiento. La primera es jTessBoxEditor, que toma como entrada una imagen TIFF (formato nativo de Tesseract) y genera un *box file*. Se trata de un archivo que contiene el tipo y la posición de un carácter en la imagen de entrenamiento. Por ejemplo, la 5ª fila contiene caracteres 4, por lo que las entradas del *box file* serán:

    - 4 20 1474 35 1500 0
    - 4 50 1471 65 1500 0
    - 4 80 1474 95 1500 0

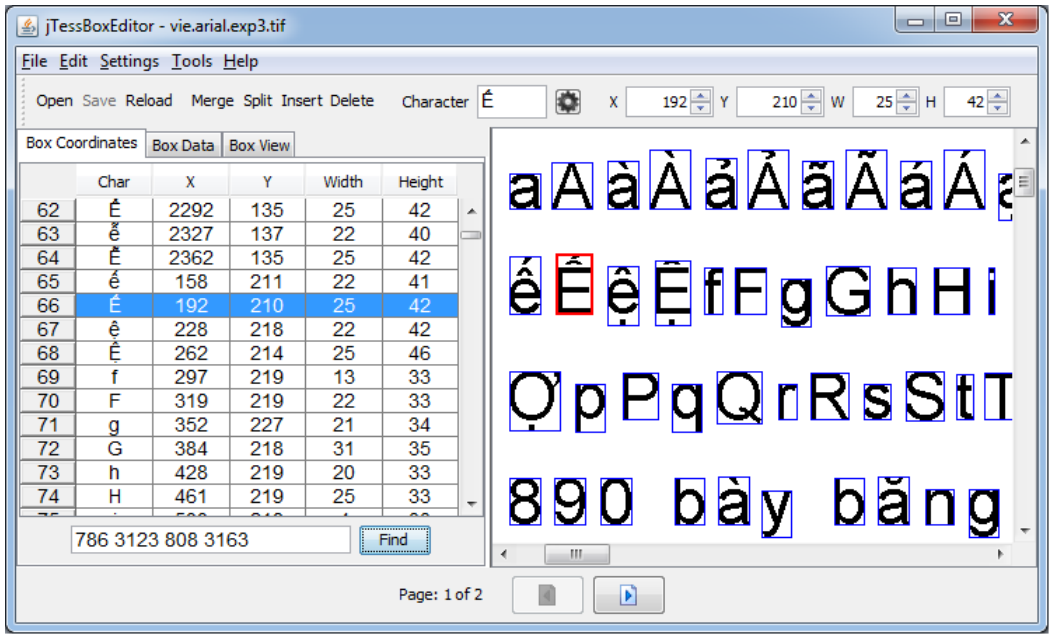

Figura 37: jTessBoxEditor editando box file

4. El paso final del proceso de entrenamiento ha sido simplificado enormemente gracias a conjuntos de scripts que forman parte de Serak Tesseract Trainer. Para obtener el archivo de entrenamiento resultante basta con usar la imagen de entrenamiento y el *box file* como parámetros de entrada de Serak.

5. Finalmente se obtiene un compacto (y por ello rápido de cargar) archivo de entrenamiento que puede llegar a aumentar significativamente la tasa de reconocimientos exitosos, siempre y cuando el entrenamiento se haya realizado de forma correcta.

Las imágenes de entrenamiento pueden formar archivos multipágina TIFF, cada una con su *box file* correspondiente y de esta forma producir archivos de entrenamiento orientados a múltiples fuentes o lenguajes, siempre y cuando sean variaciones de la misma familia.

4. Análisis y conclusiones

Una vez terminada la implementación de este proyecto, llegó la hora de evaluarlo en un entorno real y comparar, al menos, ciertas características que comparte con otro tipo de implementaciones de reconocimiento de matrículas, dejando de lado la capacidad de ser móvil, pues esta únicamente resulta útil en un sistema robusto y confiable. Aunque actualmente se trata de un tema que despierta gran interés entre muchos sectores profesionales interesados, la oferta visible al público sigue siendo baja. Es decir, a pesar de existir cierto número de empresas dedicadas a ofrecer un software de características similares, estas ofrecen las licencias a los interesados o compradores bajo NDA (*non-disclosure agreement*). Es decir, la empresa que usa la tecnología en cuestión no tiene derecho a comentar o enseñar las funcionalidades o capacidades del mismo. Esto hace la comparativas directas muy difíciles, pues no se trata de un software disponible para un público amplio.

Sin embargo, en el mercado existe una aplicación Android comercial cuyas capacidades son bien conocidas y anunciadas por la empresa que la desarrolla. De hecho la versión demo de esta aplicación sirvió como inspiración para la implementación del motor de reconocimiento de matrículas de este proyecto. Se trata de una aplicación híbrida, capaz de funcionar en cualquier tipo de dispositivo Android mínimamente moderno y cuya velocidad y precisión a la hora de reconocer matriculas de todas los formatos posibles (todas las europeas y muchas extracomunitarias) hacen de esta aplicación el líder en este segmento. Quizás los puntos más interesantes sean los hechos de que los ingenieros detrás de este proyecto no hacen uso de OpenCV (sino de una serie de algoritmos propios), además del prescindir de Tesseract (información obtenida desensamblando las librerías nativas del proyecto).

En caso de comparativa directa entre Opticar y Matrix Pocket [5], las áreas donde gana el último es en la distancia de reconocimiento exitoso de la matrícula. Opticar requiere que la matricula ocupe una parte significativa de la pantalla para producir una segmentación aceptable, mientras que Matrix Pocket puede segmentar una matrícula a varios metros de distancia muy rápidamente. Otra diferencia entre las aplicaciones, es la capacidad de reconocer matriculas extranjeras. El motor OCR de Opticar, Tesseract, ha sido entrenado con caracteres de matrículas nacionales, mientras que Matrix Pocket prescinde de un motor OCR y lo sustituye por un algoritmo de *pattern-matching* de caracteres que forman la matricula sobre la imagen obtenida, por lo que su rendimiento y precisión a la hora de aplicar el reconocimiento de caracteres a partir del segmento obtenido es muy superior.

Sin embargo, Opticar es un proyecto muy reciente y aun en estado de desarrollo activo. En un futuro, tras estudiar los fallos existentes y corregir la implementación y los algoritmos de segmentación y *subsegmentacion*, así como usando un archivo de entrenamiento adecuado, es posible que el rendimiento se acerque mucho al líder del mercado actual.

5. Bibliografía

[1] Lenguaje DRAKON - http://drakon-editor.sourceforge.net/language.html

[2] Mastering OpenCV with Practical Computer Vision Projects, Daniel Lélis Baggio, David Millán Escrivá, Packt Publishing, 2012 – Chapter 5 Number Plate Recognition Using SVM and Neural Networks

[3] Opencv 3 dev - Scene Text Detection - Class-specific Extremal Regions for Scene Text Detection http://docs.opencv.org/3.0-beta/modules/text/doc/erfilter.html

[4] Real-Time Scene Text Localization and Recognition, Lukas Neumann, Jiri Matas Centre for Machine Perception, Department of Cybernetics, Czech Technical University, Prague, Czech Republic, 2012

[5] Matrix Pocket Android - http://www.anpr.hu/matrix_pocket_android/

Anexo I – JNI

JNI es el mecanismo que permite la comunicación entre Java y C++. Para poder usarlo se requieren las siguientes partes:

1. Librería dinámica con sus métodos expuestos ya compilada e incluida en el proyecto.

2. Las librerías se tienen que cargar en el orden especifico respecto a sus dependencias

3. Los métodos a usar son marcados con la palabra clave "*native*"

La siguiente clase muestra que las librerías nativas se cargan desde un contexto estático y los prototipos de los métodos nativos únicamente se declaran.

```
public class NativeWrapper implements Native {

static {

   System.loadLibrary("lept");

   System.loadLibrary("tess");

   System.loadLibrary("opencv_java");

   System.loadLibrary("opticar");

 }


  public native String returnPlate(long matAddr);

  public native boolean initEngine();

}
```

A continuación se describe como se instancia la clase *NativeWrapper*

```
public interface Native {

  String returnPlate(long marAddrGr);
  boolean initEngine();

      public static class Factory {
       static NativeWrapper instance;
       public synchronized static Native create(){
        if(instance == null){
          instance = new NativeWrapper();
        }
       return instance;
     }
  }
}
```

Se instancia:

```
private Native nativePart;
...
if(nativePart == null){
            nativePart = Native.Factory.create();
}
```

Se llama a método nativo pasándole un valor:

```
String matricula = nativePart.returnPlate(frameAddr.longValue());
```

Para poder crear una librería nativa hay que crear los métodos a los que se quiera llamar. Nótese que los nombres tienen que ser los mismos que en Java, Por ejemplo al método de abajo no se le pasa ningún parámetro y este devuelve un *boolean*.

```
jboolean initEngine(JNIEnv* env, jobject thiz){
   ...
   return true;
}
```

A este método se le pasa un *long* y se devuelve un *String*

```
jstring returnPlate(JNIEnv* env, jobject thiz, jlong value) {
   ...
   return env->NewStringUTF("str");
}
```

A continuación se procede a exponer los métodos nativos para que estos puedan ser llamados desde Java. Esto se hace creando un *array* de tipo *JNINativeMethod* que tiene que contener los siguientes elementos:

1. nombre del método
2. signatura del método
3. (void* nombre del método)

Bajo la signatura del método se entienden los parámetros que se le pasan al método y que este devuelve, Por ejemplo, (J)LJava/lang/String; quiere decir que el método toma como parámetro un *long* (representado como (J)) y devuelve un *String* (representado como LJava/lang/String). Otro ejemplo es initEngine que no toma ningún parámetro y devuelve un *boolean* (representado cono Z).

```
static JNINativeMethod exposedMethods[] = {
 {"returnPlate","(J)Ljava/lang/String;",(void*)returnPlate},
 {"initEngine","()Z",(void*)initEngine}
};
```

Para poder registrar los métodos expuestos, se utiliza el siguiente método, que es llamado cuando la librería es cargada, En él se tienen que indicar la clase java que declara los métodos nativos (la misma que la de arriba) y el *array* que contiene los nombres y signaturas.

```
jint JNI_OnLoad(JavaVM* vm, void* reserved)
{
    JNIEnv* env;
  if (vm->GetEnv(reinterpret_cast<void**>(&env), JNI_VERSION_1_6) != JNI_OK) {
            return JNI_ERR;
 }
   jclass clazz = env->FindClass("es/sdci/opticar/NativeWrapper");
    if(clazz==NULL){
      return JNI_ERR;
    }
env->RegisterNatives(clazz, exposedMethods, sizeof(exposedMethods) / sizeof(JNINativeMethod));
    env->DeleteLocalRef(clazz);
    return JNI_VERSION_1_6;
}
```